Linear-Space Data Structure for Parametrized Range Mode Query in Arrays*

Ovidiu Rața[1], Paul Diac[1]

Alexandru Ioan Cuza Universy of Iași,
Faculty of Computer Science, Iași, Romania

Abstract. We present a data structure for the range mode query problem, studied by Chan et al [1], He,Liu.[3] and Xu, Williams et al[2], which requires $O(n)$ space, and answers queries on an interval $[i, j]$ in parametrized $O(\sqrt{j-i+1})$ time. The standard RAM model is assumed, with word size $w = \Omega(\log n)$.

Additionally, we present a linear-space data structure, that requires $O(\min(\sqrt{j-i+1}, \sqrt{j/w}))$ parametrized time per query, and supports element insertion at the end of the array $A[1:n]$ that requires $O(\sqrt{n \cdot w})$ time, by improving over the method proposed by Chan et al[1], using compact rank/select data structures that also support adding an element at the end of the binary array.

Keywords: Data structure · Parametrized algorithms · Range queries · Mode.

1 Introduction

2 Finding a Range Mode

Our data structure is constructed by extending the ideas of Chan et al[1].

Data Structure precomputation. Let D denote the set of elements of the array A, and assume some arbitrary ordering on the elements. We denote the number of distinct elements of A as Δ. First, we apply rank-space reduction, and construct the array B, such that, for each i, $B[i]$ stores the rank of $A[i]$ in D. Thus, $B[i] \in \{1, \dots, \Delta\}$. Let C be an array of size Δ, that provides a direct mapping from $\{1, \dots, \Delta\}$ to D. Set D and arrays B and C can be computed inf $O(n \log \Delta)$ time. Further, we will focus on finding a range mode x, over array B, which will be transformed into the respective range mode over array A, using array C, that is, $y = C[x]$. For each $a \in \{1, \dots \Delta\}$, let $Q_a = \{b \mid B[b] = a\}$. We will represent the sets Q_a as ordered arrays. Also, we define the array Q^{-1}, of size n, s.t. $\forall i \in \{1, \dots, n\}, Q_i^{-1} = k, Q_{B_i}(k) = i$.

* Supported by Alexandru Ioan Cuza Universy of Iași.

Additional Definitions. Let $\phi(i, j)$ be the function that returns the frequency of the most frequent element in the range $B[i : j]$.

Let $\textbf{freq}_x(i, j)$, be the frequency of element x of array B, inside the interval $[i, j]$.

Further, we will show how to build a data structure, that supports parametrized range mode query in time $O(\sqrt{j - i + 1})$ for a query interval $[i, j]$.

Lemma 1. *Given an array $A[1 : n]$, there exists a data structure, requiring arrays B, C, Q, Q^{-1} and $O(n/w)$ additional words of RAM, that for some fixed integer $L \in \{0, \cdots, \lceil \log n \rceil\}$, for intervals $[i, j]$ s.t. $2^{L-1} < j - i + 1 \leq 2^L$ supports queries of the following form, in $O(\sqrt{2^L})$ time:*

For the interval $[i, j]$, determine an element x, s.t. $\textbf{\textit{freq}}_x(i, j) \geq \min(\phi(i, j), \sqrt{2^L})$

Proof. First, we fix an integer $L \in \{0, \cdots, \lceil \log n \rceil\}$. Then, we separate the array $A[1 : n]$ into adjacent blocks, of length $b_L = \sqrt{2^L}$. The block $i \in \{1, \ldots, \lfloor \frac{n}{b_L} \rfloor\}$ encompasses the interval $[(i - 1) \cdot b_L + 1, \min(i \cdot b_L, n)]$.

Let s_i^L, be a binary string, similar to the string defined in the data structure proposed by Chan et al.[1]:

Definition 1. *For integer $L \in \{0, \cdots, \lceil \log n \rceil\}$, and block i of size $\sqrt{2^L}$, let s_i^L be a binary string, obeying the following properties:*

Property 1. Consider the interval of blocks, from block i to block $j = \min(\lfloor \frac{n}{b_L} \rfloor, i + \sqrt{2^L} - 1)$. There will be $j - i + 1 = O(\sqrt{2^L})$ bits with value 1 in s_i^L, the k-th set bit corresponding to the right border of the interval k.

Property 2. For any integer $k \in [i, j]$, consider the interval of blocks $[i, k]$.

For the interval of blocks $[i, k]$, consider the endpoints of array A which correspond to the left endpoint of the block i, and the right endpoint of the block k to be $i' = (i - 1) \cdot b_L + 1, k' = k \cdot b_L$.

In the string s_i^L, there are exactly $\min(\sqrt{2^L}, \phi(i', k'))$ bits with value 0 to the left of the k-th set bit.

Note that the length of the binary string s_i^L is at most $2 \cdot \sqrt{2^L} = O(\sqrt{2^L})$.

Every string s_i^L, can be represented with a succint or compact data structure that supports rank-select operations, in $O(\sqrt{2^L}/w)$ RAM words.

There are $\lfloor \frac{n}{2^L} \rfloor$ blocks of length b_L, thus, the space necessary for maintaining the strings s_i^L is:

$$\lfloor \frac{n}{2^L} \rfloor \cdot O(\sqrt{2^L}/w) = O(n/w) \text{ words of RAM.}$$

Query Algorithm. The query alorithm is similar to that of the data structure proposed by Chan et al.[1]:

Given the interval $[i, j]$:

1. If $j - i + 1 \leq 2 \cdot \sqrt{2^L}$, then just iterate through the elements of $[i, j]$ in $O(\sqrt{2^L})$ time, or determine the blocks that are entirely contained inside of $[i, j]$. Let the block and the last blocks be β_1 and β_2 respectively.
2. There will be at most $\sqrt{2^L}$ blocks from block β_1 to block β_2.
   Let $[i', j']$ be the interval of A, which corresponds to blocks β_1 through β_2. We will use the string $s^L_{\beta_1}$ to get the frequency f, of an element from $[i', j']$ s.t. $f \geq \min(\phi(i', j'), \sqrt{2^L})$. This can be done by select operations over $s^L_{\beta_1}$, more precisely, we will determine $p = \mathbf{select}_1(s^L_{\beta_1}, \beta_2 - \beta_1 + 1)$, which is the position in the string $s^L_{\beta_1}$, corresponding to the right endpoint of the block β_2, with respect to block β_1. The value f, corresponding to the number of bits with value 0 in the interval $[1, p_2]$ of the string $s^L_{\beta_1}$, can be calculated as follows:
   $$f = (p_2) - (\beta_2 - \beta_1 + 1)$$
   We can further use binary search, to get the number β_f and the position p_f in string $s^L_{\beta_1}$ of the first block, such that there are exactly f bits with value 0 in the interval $[1, p_f]$ of the string $s^L_{\beta_1}$. Further, we can iterate through each position k, of the block β_f, and use arrays Q_{B_k} in order to determine an element x with frequency at least f. (Chan et al.[1])
   This procedure will take $O(\sqrt{2^L})$ time, as the most time consuming step is the iteration through elements of β_f.
   We must also note, that this step will yield exactly the answer for the query $[i', j']$.
3. Further, we can iterate through each position $k \in ([i, j] \setminus [i', j'])$, and increase f every time we can, by checking whether $Q_{B_k}(Q^{-1}_k \pm (f + 1)) \in [i, j]$. This will clearly take $O(\sqrt{2^L})$ time, as there will be $O(\sqrt{2^L})$ elements in the prefix and suffix of $[i, j]$, and we will increase f at most $O(\sqrt{2^L})$.
   This step clearly determines the answer for the query $[i, j]$, as it uses the value f characterising the answer for the query (i', j'), and uses the prefix and suffix of $[i, j]$, in order to adapt f to the answer for the query (i, j). These steps are the same as in the data structure proposed by Chan et al.[1], and proofs for their correctness are also provided in their work.

Finally, the required element x of the range $B[i : j]$, can be transformed to the corresponding element of A, which is C_x. Thus, the answer to the query (i, j) given above, is calculated in $O(\sqrt{2^L})$ time.□

Further, for any integer $L \in \{0, \cdots, \lceil \log n \rceil\}$ we will define $b'_L = 2^L$, as the size of the big blocks, and will separate the array $B[1 : n]$ into adjacent blocks of size b'_L. If we cannot exactly divide the array $B[1 : n]$ into blocks of size b'_L, then the last elements of array $B[1 : n]$ will just be included into a last block, of size $< b'_L$.

For every block $i \in \{1, \ldots, \lceil \frac{n}{b'_L} \rceil\}$ of size b'_L, we denote by F^L_i, the set of elements in the range $B[(i-1) \cdot b'_L + 1 : \min((i+1) \cdot b'_L, n)]$, that have frequency $> \sqrt{2^L}$. Note that the interval $[(i-1) \cdot b'_L + 1 : \min((i+1) \cdot b'_L, n)]$ contains 2 big blocks, if its size is not limited by n.

If we maintain the sets F_i^L as ordered arrays, then the amount of space required for storing F_i^L will be:

$$\sum_{L=0}^{\lceil \log n \rceil} \sum_{i=1}^{\lceil \frac{n}{b'_L} \rceil} |F_i^L| \leq \sum_{L=0}^{\lceil \log n \rceil} \sum_{i=1}^{\lceil 2 \cdot \frac{n}{b'_L} \rceil} 2 \cdot \sqrt{2^L}$$

$$= O(\sum_{L=0}^{\inf} \sum_{i=1}^{\lfloor \frac{n}{2^L} \rfloor} \sqrt{2^L}) = O(\sum_{L=0}^{\inf} \lfloor \frac{n}{2^L} \rfloor \cdot \sqrt{2^L})$$

$$= O(\sum_{L=0}^{\inf} \frac{n}{\sqrt{2^L}}) = O(\sum_{L=0}^{\inf} \frac{n}{\sqrt{2^L}}) = O(n \cdot \sum_{L=0}^{\inf} \frac{1}{\sqrt{2^L}})$$

$$= O(n \cdot \frac{\sqrt{2}}{\sqrt{2} - 1}) = O(n)$$

Thus, storing the sets F_i^L is $O(n)$ words of RAM, as storing each element of F_i^L requires 1 word of RAM.

Lemma 2. *Given an array $A[1 : n]$, there exists a data structure, requiring arrays B, C, Q, Q^{-1}, F_i^L and $O(n/w)$ additional words of RAM, that for some fixed integer $L \in \{0, \cdots, \lceil \log n \rceil\}$, for intervals $[i, j]$ s.t. $2^{L-1} \leq j - i + 1 < 2^L$ and supports queries of the following form, in $O(\sqrt{2^L})$ time:*

For the range $A[i : j]$, determine an element of frequency $\phi(i, j)$, if $\phi(i, j) > \sqrt{2^L}$, or -1 if no such element exists.

Proof. First, we fix an integer $L \in \{0, \cdots, \lceil \log n \rceil\}$. Then, we separate the array $A[1 : n]$ into adjacent blocks, of length $b'_L = 2^L$. The block $i \in \{1, \ldots, \lceil \frac{n}{b'_L} \rceil\}$ encompasses the interval $[(i-1) \cdot b'_L + 1, \min(i \cdot b'_L, n)]$.

For each element x in F_i^L, we define the binary string $Y_{i,x}^L$, the following way:

Definition 2. *For integer $l \in \{0, \cdots, \lceil \log n \rceil\}$, for each block i of size 2^L, for each element $x \in F_i^L$, let $Y_{i,x}^L$ be a binary string, obeying the following properties:*

*Property 3. The string $Y_{i,x}^L$ will have exactly $2 \cdot \sqrt{2^L}$ bits set to value 1. Each of these bits, will correspond to the right endpoint of a small block of size $b_L = \sqrt{2^L}$, declared in **lemma 1**, that lies inside the big block i, of size b'_L, or in the big block $i + 1$.*

Property 4. For every k, from 1 to $2 \cdot \sqrt{2^L}$, let $k' = \min(k' \cdot b_L + i', n)$ be the right endpoint of the k-th small block, counting from the first small block inside of the big block i, or just n, if such a block does not exist. Note that the small block k may lie inside the big block $i + 1$.

Inside the string $Y_{i,x}^L$, there will be exactly $\boldsymbol{freq}_x(i', k')$ bits with value 0 to the left of the k-th bit with value 1.

For string $Y^L_{i,x}$, there will be exactly $\mathbf{size}_0 = \mathbf{freq}_x(b'_L \cdot (i-1)+1, \min(b'_L \cdot (i+1), n))$ bits with value 0. Note that $\mathbf{size}_0 > \sqrt{2^L}$. Also, there will be exactly $\mathbf{size}_1 = 2 \cdot \sqrt{2^L}$ bits with value 1. Thus, the size of the string will be : $\mathbf{size}_0 + \mathbf{size}_1 \leq 3 \cdot \mathbf{size}_0$.

Thus, the total length of the strings $Y^L_{i,x}$, over the blocks $i \in \{1, \ldots, \lceil \frac{n}{b'_L} \rceil\}$ will be:

$$\sum_{i=1}^{\lceil \frac{n}{b'_L} \rceil} \sum_{x \in F^L_i} |Y^L_{i,x}| \leq \sum_{i=1}^{\lceil \frac{n}{b'_L} \rceil} \sum_{x \in F^L_i} 3 \cdot \mathbf{size}_0$$

$$= \sum_{i=1}^{\lceil \frac{n}{b'_L} \rceil} \sum_{x \in F^L_i} 3 \cdot \mathbf{freq}_x(b'_L \cdot (i-1)+1, \min(b'_L \cdot (i+1), n))$$

$$\leq \sum_{x \in F^L_i} 3 \cdot 2 \cdot \mathbf{freq}_x(1, n) \leq \sum_{x \in B[1:n]} 6 \cdot \mathbf{freq}_x(1, n) = O(n)$$

The total $Y^L_{i,x}$ length of the strings will be $O(n)$ bits, thus, for a fixed L, they will require $O(n/w)$ words of RAM to be stored, and to support rank/select operations in $O(1)$.

Query Algorithm. The query alorithm is as follows:

Given the interval $[i, j]$:

1. If $j - i + 1 \leq 2 \cdot \sqrt{2^L}$, then just iterate through the positions of $B[i, j]$ and determine the answer. Otherwise, the interval $[i, j]$ will intersect, at most 2 big blocks of size b'_L. Let these blocks be β'_1 and β'_2. Let the interval of small blocks that are entirely covered by the interval $[i, j]$, be $[\beta_1, \beta_2]$. These blocks can be determined rapidly, as $\beta_2 = \lfloor j/b_L \rfloor$ and $\beta_1 = \lfloor (i + b_L)/b_L \rfloor$, and similarly for the big blocks β'_1 and β'_2.
2. Further, we iterate through each element $x \in F^L_{\beta'_1}$, and determine :

$$p^x_1 = \mathbf{select}_1(Y^L_{\beta'_1,x}, \beta_1 - (\beta'_1 - 1) \cdot \frac{b'_L}{b_L} - 1), p^x_2 = \mathbf{select}_1(Y^L_{\beta'_1,x}, \beta_2 - (\beta'_1 - 1) \cdot \frac{b'_L}{b_L})$$

$$f_x = (p^x_2 - p^x_1) - (\beta_2 - \beta_1 + 1), f = \max_{x \in F^L_{\beta'_1}} (f_x)$$

3. Now, we iterate through the positions k of the prefix and the suffix of the interval $[i, j]$, which are not contained in any of the small blocks in the interval$[\beta_1, \beta_2]$, and check wether we can increase f by 1 or not, by verifying if $Q_{B_k}(Q^{-1}_k \pm (f+1)) \in [i, j]$.
4. Finally, if f will achieve a value $> \sqrt{2^L}$, then return f, or -1 otherwise.

We can clearly see, that the runtime of the query algorithm will be $O(\sqrt{2^L})$, as the first step requires $O(1)$ time, the second step requires $O(\sqrt{2^L})$ time, as at most $O(\sqrt{2^L})$ elements of the set F_i^L will be verified and the **select** operations will take $O(1)$ time. The third step will also take $O(\sqrt{2^L})$ time, as there will be $O(\sqrt{2^L})$ elements of the prefix and the suffix which will have to be verified, and the value of f will be increases by 1 at most $O(\sqrt{2^L})$ times. Note that the element x with maximum frequency, can also be easily tracked during each step.

Further, we will use the results from **lemma 1** and **lemma 2** to prove that a $O(\sqrt{j-i})$ runtime per query is possible, by using $O(n)$ space.

Theorem 1. *Given an array $A[1:n]$, there exists a data structure, requiring arrays B, C, Q, and additional $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in time $O(\sqrt{j-i+1})$.*

Proof. Firstly, we will build the arrays B, C, Q, and Q^{-1}, which will take $O(n \log n)$ time and $O(n)$ space. Also, for each $L \in \{0, \dots, \lceil \log n \rceil\}$, we will build F_i^L. It is easy to see, that this will take $O(n \log n)$ time, and $O(n)$ space.

Further, for each $L \in \{0, \dots, \lceil \log n \rceil\}$, we will build an instance of the data structure described in **lemma 1**, denoted by DS_1^L, and an instance of the data structure described in **lemma 2**, denoted by DS_2^L. These instances will take $O(n)$ space, as the arrays B, C, Q, Q^{-1} and F_i^L will be shared among them, and the additional information will take $O(\log n \cdot n/w) = O(n)$ space.

Query Algorithm. The query algorithm is as follows:

Given the interval $[i,j]$:

1. Determine the smallest L, s.t $2^{L-1} < (j-i+1) \leq 2^L$. Note that $2^L = O(j-i+1)$.
2. We will determine $(f,x) = \max(DS_1^L.query(i,j), DS_2^L.query(i,j))$, where $DS_1^L.query(i,j)$, and $DS_2^L.query(i,j)$ denotes querying the instances DS_1^L and DS_2^L, respectively, for the interval $[i,j]$. (f,x) will be exactly the answer for the query (i,j), as $f = \phi(i,j)$, and x is an element of the array A, s.t. $\exists y, s.t.\ (C_y = x) \wedge (\mathbf{freq}_y(i,j) = f)$.

The algorithm will take $O(\sqrt{2^L}) = O(\sqrt{j-i+1})$ time, as $2^L = O(j-i+1)$□.

Lemma 3. *Given an array $A[1:n]$, there exists a data structure requiring $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in $O(\sqrt{j/w})$ time.*

Proof. Firstly, we build the arrays B, C, Q, Q^{-1}, as in the previous data structures.

In their work, Chan et al.[1] have divided the array $A[1:n]$ into blocks of equal length, in order to prove their result. We will take a similar approach, one of the differences being, that we will divide the array $A[1:n]$, into $O(\sqrt{w \cdot n})$ blocks of varying size.

Firstly, we divide the array $A[1:n]$ into adjacent big blocks, block i (counting from left to right) having size i. It is easy to see, that there will be at most $O(\sqrt{n})$ big blocks. Further, we will divide each big block i, into small blocks, of size $i/\sqrt{w}$. Let the number of small blocks be b. For each small block β, of size $size(\beta)$, we will store the string z_β and the set H_β, defined below:

Definition 3. *For each small block $\beta \in \{1,\ldots,b\}$, let z_β be a binary string obeying the following properties:*

Property 5. There are exactly β bits with value 1 *in string z_β. The k-th bit with value* 1 *counting from left to right will correspond to the left endpoint of the k-th small interval.*

Property 6. For each bit k with value 1*, let l_k be the position of the left endpoint of the small block k in array B, and let r_β be the position of the right endpoint of the small block β in array B. There will be exactly* $\min(\phi(l_k, r_\beta), b)$ *bits with value* 0*, to the right of the k-th bit with value* 1*.*

Definition 4. *For each small block $\beta \in \{1,\ldots,b\}$, having the right endpoint at position r_β of the array B, let H_β be the set of elements $x \in B$, s.t. $\boldsymbol{freq}_x(1,r_\beta) > \beta$.*

We must note that, $|H_\beta| < r_\beta/\beta$, thus, $|H_\beta| = O(\sqrt{r_\beta/w})$.

Further, for each element x, which is present in at least one set H_β, we will define the string η_x the following way:

Definition 5. *For each element x present in at least one set H_β, let β_{max} be rightmost block, s.t. $x \in H_{\beta_{max}}$. Let η_x be a binary string, obeying the following properties:*

Property 7. There are exactly β_{max} bits with value 1 *in string η_x, k-th bit with value* 1 *corresponding to the right endpoint of the k-th small block.*

Property 8. For each $k \in \{1,\ldots,\beta_{max}\}$, let r_k be the position of the right endpoint of the k-th small block in array B. There are exactly $\boldsymbol{freq}_x(1,r_k)$ *bits with value* 0 *to the left of the k-th bit with value* 1 *of string η_x.*

For each block β, we can store each set H_β as an ordered array of pairs of integers (x,p_x), x representing the element of H_β which is being stored, and p_x is a pointer to the data structure storing the string η_x.

As, for each small block β, $|z_{beta}| = O(\sqrt{n \cdot w})$ and $b = O(\sqrt{n \cdot w})$ the total space needed to store the strings z_β will be:

$$\sum_{\beta=1}^{b} |z_\beta| = O(\sqrt{n \cdot w}) \cdot O(\sqrt{n \cdot w}) = O(n \cdot w) \text{ bits}$$

$O(n \cdot w)$ bits will be needed to store the strings z_β, thus, $O(n)$ words of RAM will be stored.

As for each small block β, $|H_\beta| = O(\sqrt{n/w})$, the space needed to store the sets H_β will be:

$$\sum_{i=1}^{b} |H_i| = O(\sqrt{n/w}) \cdot O(\sqrt{n \cdot w}) = O(n) \text{ words of RAM.}$$

For each string η_x, let $\beta_{\max}$ be the rightmost small block that contains x, let the number of bits with value 0, be $size_0(\eta_x)$, and the number of bits with value 1 be $size_1(\eta_x)$.

$$size_0(\eta_x) > \beta_{\max}, size_1(\eta_x) = \beta_{\max} \implies size_0(\eta_x) > size_1(\eta_x)$$

Thus, the space required to store the strings η_x will be:

$$\sum_{x \in (\cup_{\beta=1}^{b} H_\beta)} |\eta_x| = \sum_{x \in (\cup_{\beta=1}^{b} H_\beta)} size_0(\eta_x) + size_1(\eta_x)$$

$$\leq \sum_{x \in (\cup_{\beta=1}^{b} H_\beta)} 2 \cdot size_0 \eta_x \leq \sum_{x \in (\cup_{\beta=1}^{b} H_\beta)} \mathbf{freq}_x(1, n)$$

$$= O(n) \text{ bits of space.}$$

Thus, for maintaining the strings η_x, we need $O(n/w) = o(n)$ words of RAM.

Finally, we may state that the total space required by this data structure is $O(n)$.

Query Algorithm. The query algorithm is as follows:

Given the interval $[i, j]$:

1. We determine the rightmost big block β', of size β', which intersects $[i, j]$. As the length of the prefix of B, covered with big blocks 1 through β', is :

$$\sum_{k=1}^{\beta'} k = \frac{\beta' \cdot (\beta' + 1)}{2} \leq 2 \cdot j$$

We can imply that:

$$\beta'^2 \leq 2 \cdot (\frac{\beta' \cdot (\beta' + 1)}{2}) \leq 4 \cdot j \implies \beta' = O(\sqrt{j})$$

Afterwards, we determine the leftmost and the rightmost small blocks, β_1 and β_2, respectively, that are fully contained in the interval $[i, j]$. Note that:

$$size(\beta_1) \leq size(\beta_2)$$

$$size(\beta_2) = \beta' / \sqrt{w} = O(\sqrt{j/w})$$

If there are no small blocks that are fully contained in the interval $[i, j]$, then we can just iterate through all the elements of $[i, j]$ and determine the range mode, which will take $O(\sqrt{j/w})$ time.

2. Let l_1 be the left endpoint of the small block β_1 and r_2 be the right endpoint of the small block β_2 in array B. We query the string z_{β_2}, and get the values $p_2 = |z_{\beta_2}|$, and $p_1 = \mathbf{select}_1(z_{\beta_2}, \beta_1)$ corresponding to the left endpoint of the small block β_1. Further, we calculate:
$$f = (p_2 - p_1) - (\beta_2 - \beta_1)$$
which corresponds to $f = \min(\phi(l_1, r_2), \beta_2)$.
In order to get an element x, which has frequency at least f in the interval of small blocks $[\beta_1, \beta_2]$, we can get the rightmost position p', corresponding to the left end of the small block β', such that, there are exactly f bits with value 0 in the interval $[p', p_2]$ in the string η_x. This can be done easily, using **rank** and **select** operations, as stated by Chan et al.[1]. Then, we need to iterate through each position k of the small block β', and check whether there are at least f occurrences of B_k inside the interval of small blocks $[k, r_2]$.
3. Further, we iterate through the elements x of H_{β_2+1}, and find the values of their frequencies in the interval of small blocks $[\beta_1, \beta_2]$. In order to find the frequency of element x in the interval of small blocks $[\beta_1, \beta_2]$, we query the string η_x for values:
$$p_1 = \mathbf{select}_1(\eta_x, \beta_1 - 1)$$
$$p_2 = \mathbf{select}_1(\eta_x, \beta_2)$$
p_1 being the position of the bit with value 1 in string η_x, corresponding to the right endpoint of the small interval $\beta_1 - 1$, and p_2, being the position of the bit with value 1, corresponding to the right endpoint of the small interval β_2. Thus, the frequency of the element x will be:
$$f_x = (p_2 - p_1) - (\beta_2 - \beta_1 + 1)$$
Further, we update the value of f the following way:
$$f := \max(f, \max_{x \in H_{\beta_2+1}} (f_x))$$
This way, in a manner similar to that presented in **lemma 1** and **lemma 2**, we obtain the value $\phi(l_1, r_2)$, which is exactly the frequency of a mode, in the interval corresponding to the small blocks $\{\beta_1, \beta_1 + 1, \ldots, \beta_2\}$. We must note, that during this step, it is also easy to keep track of an element x of maximum frequency.
4. Now, we must iterate through positions $k \in ([i, j] \setminus [l_1, r_2])$ and try to increase the frequency f, using the arrays Q, B and Q^{-1}, as in **lemma 1** and **lemma 2**. This will take at most $O(\sqrt{j/w})$ time, which is an upper bound on the sizes of the small blocks β_1 and β_2.

This query algorithm will clearly run in time $O(\sqrt{j/w})$, as step 2 takes at most $O(\sqrt{j/w})$ time, step 3 takes at most $O(\sqrt{r_2/w}) = O(\sqrt{j/w})$ time, and step 4 similarly, takes at most $O(\sqrt{j/w})$ time. □

Theorem 2. *Given an array $A[1:n]$, there exists a data structure, requiring $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in time $O(\min(\sqrt{j-i+1}, \sqrt{j/w}))$.*

Proof. Firstly, we will build the arrays B, C, Q, and Q^{-1}, which will take $O(n\log n)$ time and $O(n)$ space. Further, we will build an instance of the data structure defined in **lemma 3**, denoted by DS_1, and an instance of the data structure defined in **theorem 1**, denoted by DS_2. The arrays, as well as both of the instances of the data structures mentioned above, will require $O(n)$ space, thus, the space required by the current data structure is $O(n)$.

Query Algorithm. The query algorithm is as follows:

Given the interval $[i,j]$:

1. Estimate the time required by a query over the data structure DS_1 and the interval $[i,j]$. Let this time estimation be t_1. Estimate the time required by a query over the data structure DS_2 and the interval $[i,j]$. Let this time estimation be t_2.
2. If $t_1 < t_2$, then, return $DS_1.query(i,j)$, otherwise, return $DS_2.query(i,j)$.

It is easy to do the time estimates t_1 and t_2 up to a constant factor, in at most logarithmic time.

Thus, the runtime of the query algorithm is $O(\min(\sqrt{j-i+1}, \sqrt{j/w}))$.□

3 Adding Elements at the End of the Array

Further, considering the fact, that we can implement a compact, $O(1)$ rank/select, data structure, that support adding a bit to the end of the array in $O(1)$, we will show how to make the data structure from section 2 support adding an element at the end of the array $A[1:n]$ in $O(\sqrt{n\cdot w})$ time, while requiring linear space and keeping the same query time.

Lemma 4. *Given an array $A[1:n]$, there exists a data structure, that respects the same conditions as the data structure constructed in Lemma 1, and also supports adding an element at the end of the array $A[1:n]$, in $O(\sqrt{2^L})$ time.*

Proof. As the only data stored in the data structure from **lemma 1**, are the strings $s^L_{\beta_1}$, we will present an algorithm to update $s^L_{\beta_1}$, as new elements are added to the end of array A.

Update Algorithm. Suppose that there are β' big blocks of size exactly $b'_L = 2^L$ that are fully contained in array $B[1:n]$. Let β be the number of small blocks, of size $b_L = \sqrt{2^L}$, that are fully contained in the array $B[1:n]$. Let $\beta_{new} = \beta+1$ be the new small block, which is constructed as new elements are appended to the end of $A[1:n]$.

An update is given, in the form of a number x, that must be added to the end of $A[1:n]$.

1. Determine the element y of array B, s.t. $C_y = x$, then add $n+1$ to the end of Q_y, and add $size(Q_y)$ to the end of Q^{-1}. If such an element y does not exist, then, $\Delta := \Delta + 1$ and $y := \Delta$, add x to the end of the array C, define $Q_y = \{n+1\}$, and add 1 to the end of Q^{-1}. Add y to the end of B.
2. For each small block $\beta_i \in \{\beta_{new} - \sqrt{2^L} + 1, \dots, \beta_{new}\}$, let l_i be the left endpoint of β_i. We must check, whether the new element y added to $B[1:n]$, changes the maximum frequency inside the interval of small blocks $[\beta_i, \beta_{new}]$, by verifying the following condition:

$$p = |s^L_{\beta_i}|, c_1 = \mathbf{rank}_1(s^L_{\beta_i}, p)$$

$$f = p - c_1, l_y = Q_y(Q^{-1}(n+1) - f - 1)$$

$$l_y \geq l_i \text{ and } f + 1 \leq \sqrt{2^L}$$

   If this condition holds, then we add a bit with value 0 to the end of $s^L_{\beta_i}$. This will signify the increase in the maximum frequency of an element, recorded by the string $s^L_{\beta_i}$. We must also note, that the strings $s^L_{\beta_i}$ will change their structure during these intermediary frequency-increase steps, as bits with value 0 will be present at the end of $s^L_{\beta_i}$. This will not influence the queries over $s^L_{\beta_i}$ required by the data structure from **lemma 1**.
3. If the position $n+1$ is the right endpoint of a new small block β_{new} that is now fully contained in array $B[1:(n+1)]$, then, for each $\beta_i \in \{\beta_{new} - \sqrt{2^L} + 1, \dots, \beta_{new}\}$, we must update the information stored in $s^L_{\beta_i}$. In such a case, we must add 1 to the end of $s^L_{\beta_i}$, which will signify the addition of a right endpoint of a new full small block.

It is easy to see that the first step takes at most logarithmic time. The second step, will take $O(\sqrt{2^L})$ time, as there may be at most $O(\sqrt{2^l})$ small blocks β_i, and each append of a bit to the end of β_i takes $O(1)$ time. Similarly, the third step takes $O(\sqrt{2^L})$ time.

The required space will still remain linear, as $s^L_{\beta_1}$ has been proven to require $O(n/w)$ words of RAM, and $O(1)$ more space will be added at each update. □

Lemma 5. *Given an array $A[1:n]$, there exists a data structure, that respects the same conditions as the data structure constructed in Lemma 2, and also supports adding an element at the end of the array $A[1:n]$, in $O(\sqrt{2^L})$ time.*

Proof. As the only data stored in the data structure from **lemma 2**, are the sets F^L_i and the strings $Y^L_{i,x}$, we will present an algorithm to update them, as new elements are added to the end of array A.

Update Algorithm. Suppose that there are β' big blocks of size exactly $b'_L = 2^L$ that are fully contained in array $B[1:n]$. Let β be the number of small blocks, of size $b_L = \sqrt{2^L}$, that are fully contained in the array $B[1:n]$. Let

$\beta_{new} = \beta + 1$ be the new small block, which is constructed as new elements are appended to the end of $A[1:n]$, and $\beta'_{new} = \beta' + 1$ be the new big block, which is constructed as new elements are appended to the end of $A[1:n]$.

An update is given, in the form of a number x, that must be added to the end of $A[1:n]$.

1. Determine the element y of array B, s.t. $C_y = x$, then add $n+1$ to the end of Q_y, and add $size(Q_y)$ to the end of Q^{-1}. If such an element y does not exist, then, $\Delta := \Delta + 1$ and $y := \Delta$, add x to the end of the array C, define $Q_y = \{n+1\}$, and add 1 to the end of Q^{-1}. Add y to the end of B.
2. We must first check, if the frequency of the element y in the interval of B, corresponding to the interval of big blocks $[\beta', \beta'_{new}]$ is $> \sqrt{2^L}$. According to Chan et al.[1], this can be done in $O(\log n)$ time, using the arrays Q_y and Q^{-1}.
   If the frequency of y is not big enough, then we can stop the update procedure.
   Otherwise, we must check whether the element y is present in $F^L_{\beta'_{new}}$. This can be done simply, by iterating through the elements of $F^L_{\beta'_{new}}$.
   If y is present in $F^L_{\beta'_{new}}$, then we must add 0 to the end of $Y^L_{\beta_{new},y}$. Otherwise, we must create the string $Y^L_{\beta_{new},y}$, and add the pair (y, p_y) to the end of $F^L_{\beta_{new}}$.
3. In this step, we will present a procedure, to create the string $Y^L_{\beta_{new},y}$, when y has to be added to the set $F^L_{\beta_{new}}$ for the first time.

Theorem 3. *Given an array $A[1:n]$, there exists a data structure, requiring arrays B, C, Q, and additional $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in $O(\sqrt{j-i+1})$ time, and supports adding an element at the end of $A[1:n]$ in $O(\sqrt{n})$ time.*

Proof.

Lemma 6. *Given an array $A[1:n]$, there exists a data structure requiring $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in $O(\sqrt{j/w})$ time, and supports adding an element at the end of $A[1:n]$ in $O(\sqrt{n \cdot w})$ time.*

Proof.

Theorem 4. *Given an array $A[1:n]$, there exists a data structure, requiring $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in $O(\min(\sqrt{j-i+1}, \sqrt{j/w}))$ time, and supports adding an element at the end of $A[1:n]$ in $O(\sqrt{n \cdot w})$ time.*

Proof.

4 Compact rank/select data structure

We present a compact data structure, that supports $O(1)$ rank/select/access operations over a binary array B of length Z, by using $O(Z)$ additional bits of space, and supports adding bits, one-by-one, at the end of array B in $O(1)$ time.

References

1. Chan, T.M., Durocher, S., Larsen, K.G., Morrison, J., Wilkinson, B.T.: Linear-space data structures for range mode query in arrays. Theory of Computing Systems **55**(4), 719–741 (2014)
2. Gu, Y., Polak, A., Williams, V.V., Xu, Y.: Faster monotone min-plus product, range mode, and single source replacement paths. arXiv preprint arXiv:2105.02806 (2021)
3. He, M., Liu, Z.: Exact and Approximate Range Mode Query Data Structures in Practice. In: Georgiadis, L. (ed.) 21st International Symposium on Experimental Algorithms (SEA 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 265, pp. 19:1–19:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.SEA.2023.19, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SEA.2023.19