Dynamic Range Mode Enumeration

Tetto Obata[1,a)]

Abstract: The range mode problem is a fundamental problem and there is a lot of work about it. There is also some work for the dynamic version of it and the enumerating version of it, but there is no previous research about the dynamic and enumerating version of it. We found an efficient algorithm for it.

Keywords: range mode query, dynamic data structure, enumeration

1. Introduction

Definition 1 (mode)**.** *A : multiset*
$a \in A$ is a mode of A
$\Leftrightarrow \forall b \in A$ (the multiplicty of a in A) $\geq$ (the multiplicty of b in A)

In the following, "a mode of multiset $\{A[l], A[l+1], \ldots, A[r]\}$" is abbreviated to "a mode of $A[l:r]$" for a sequence A.

Problem 2 (Range mode problem)**.** *Given a sequence A over an alphabet set Σ, process a sequence of queries.*

- $\mathrm{mode}(l, r)$*: output one of the modes of $A[l{:}r]$*

The range mode problem is a fundamental problem and there is a lot of work about it.

	space complexity (bits)	query time complexity	conditions
[1]	$\mathrm{O}(n^{2-2\epsilon}\log n)$	$\mathrm{O}(n^{\epsilon})$	$0 \leq \epsilon \leq \frac{1}{2}$
[2]	$\mathrm{O}(n^{2-2\epsilon})$	$\mathrm{O}(n^{\epsilon})$	$0 \leq \epsilon \leq \frac{1}{2}$
[3]	$\mathrm{O}\left(\frac{n^2 \log\log n}{\log n}\right)$	$\mathrm{O}(1)$	
[4]	$\mathrm{O}(nm\log n)$	$\mathrm{O}(\log m)$	
[5]	$\mathrm{O}((n^{1-\epsilon}m + n)\log n)$	$\mathrm{O}(n^{\epsilon} + \log\log n)$	$0 \leq \epsilon \leq \frac{1}{2}$
[6]	$\mathrm{O}\left(4^k nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}\right)$	$\mathrm{O}(2^k)$	$k \in \mathbb{Z}_{\geq 0}$
[6]	$\mathrm{O}(nm)$	$\mathrm{O}(\min(\log m, \log\log n))$	
[6]	$\mathrm{O}\left(nm\left(\log\log\frac{n}{m}\right)^2\right)$	$\mathrm{O}\left(\log\log\frac{n}{m}\right)$	

Table 1 The results of previous research about the range mode problem. n is the length of a string and m is the maximum frequency of an item. Space complexity does not include the input string.

As a natural extension of the range mode problem, we can consider the enumeration version of the problem.

Problem 3 (Range mode enumeration problem)**.** *Given a sequence A over an alphabet set Σ, process a sequence of queries.*

- $\mathrm{modes}(l, r)$*: enumerate the modes of $A[l{:}r]$*

There is another natural extension of it, the dynamic version of the problem.

Problem 4 (Dynamic range mode problem)**.** *Given a sequence A over an alphabet set Σ, process a sequence of queries of the following three types:*

- $\mathrm{insert}(c, i)$*: insert $c\,(\in \Sigma)$ so that it becomes the i-th element of A*
- $\mathrm{delete}(i)$*: delete the i-th element of A*
- $\mathrm{mode}(l, r)$*: output one of the modes of $A[l{:}r]$*

There is some work about the range mode enumeration problem and the dynamic range mode problem.

	space complexity (bits)	query time complexity	condition
[6]	$\mathrm{O}(n^{2-2\epsilon}\log n)$	$\mathrm{O}(n^{\epsilon}\,\lvert output\rvert)$	$0 \leq \epsilon \leq \frac{1}{2}$
[6]	$\mathrm{O}\left(nm\left(\log\log\frac{n}{m}\right)^2 + n\log n\right)$	$\mathrm{O}\left(\log\log\frac{n}{m} + \lvert output\rvert\right)$	
[6]	$\mathrm{O}(nm + n\log n)$	$\mathrm{O}(\log m + \lvert output\rvert)$	
[6]	$\mathrm{O}(n^{1+\epsilon}\log n + n^{2-\epsilon})$	$\mathrm{O}(\log m + n^{1-\epsilon} + \lvert output\rvert)$	$0 \leq \epsilon \leq 1$

Table 2 The results of previous research about the range mode enumeration problem. n is the length of a string and m is the maximum frequency of an item. Space complexity does not include the input string.

	space complexity(words)	query time complexity
[7]	$\mathrm{O}(n_{\max})$	$\mathrm{O}\left(n_{\max}^{\frac{2}{3}}\right)$
[8]	$\tilde{\mathrm{O}}(n^{1.327997})$	$\tilde{\mathrm{O}}(n^{0.655994})$

Table 3 The results of previous research about the dynamic range mode problem where n is the length of string and $n_{\max}$ is the limit of the length of the string. Space complexity does not include the input string. The query time comlexity is same in all the query types. The word-size is $\Omega(\log n)$.

Considering the normal version, enumerating version, and the dynamic version of the problem, we can consider another problem, the dynamic enumerating version one.

Problem 5 (Dynamic range mode enumeration problem)**.** *Given a sequence A over an alphabet set Σ, process a sequence of queries of the following three types:*

- $\mathrm{insert}(c, i)$*: insert $c\,(\in \Sigma)$ so that it becomes the i-th element of A*
- $\mathrm{delete}(i)$*: delete the i-th element of A*
- $\mathrm{modes}(l, r)$*: enumerate the modes of $A[l{:}r]$*

There is no previous research about the dynamic range

[1] Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo 113-0032, Japan
[a)] obata-tetto777@g.ecc.u-tokyo.ac.jp

mode enumeration problem.

It is known that the range mode problem is related to the boolean matrix problem and the set intersection problem [5].

Problem 6 (Set intersection problem)**.** *Given multisets $S_1, S_2, \ldots, S_N$ of a universe U, process a sequence of following queries.*

- intersect(i, j): *check whether S_i and S_j intersect or not*

If the range mode problem can be solved efficiently, it can be checked if two sets intersect efficiently. We can solve the set intersection problem by building a data structure for a sequence of $2N|U|$ elements as follows

$$\text{(elements of) } S_1, S_1^c, S_1^c, S_1, S_2, S_2^c, S_2^c, S_2, \ldots, S_N, S_N^c, S_N^c, S_N$$

and calling mode$(2i|U| - |S_i|, 2(j-1)|U| + |S_j|)$ query for a intersect(i, j) $(i < j)$ query.

Therefore if the dynamic range mode enumeration problem can be solved efficiently, the computation of the intersection of two sets and modifying of the sets can be done efficiently.

Our contribution

Existing methods for the dynamic range mode problem cannot be applied to the dynamic range mode enumeration problem. The step 3 of Algorithm 1 of [7] cannot be used for the dynamic range mode enumeration problem. Problem 7 of [8] needs only one index and the algorithm of this paper is based on this problem. In this paper, we found the first algorithm for the range dynamic enumeration problem, which can deal with insert and delete queries in $\mathrm{O}\left(N^{\frac{2}{3}} \log \sigma'\right)$ time per query and modes query in $\mathrm{O}\left(N^{\frac{2}{3}} \log \sigma' + |output|\right)$ time per query where N is the length of the sequence and $\sigma' = |\{c \in \Sigma | c \text{ appears in the sequence}\}|$.

2. Main Result

The following theorem is the main result.

Theorem 7. *There exists a data structure for the dynamic range mode enumeration problem in the word RAM model with $\Omega(\log N + \log \sigma)$ bits wordsize in $\mathrm{O}\left(N^{\frac{2}{3}} \log \sigma'\right)$ time per* insert *and* delete *query and $\mathrm{O}\left(N^{\frac{2}{3}} \log \sigma' + |output|\right)$ time per* modes *query where N is the length of the sequence and*
$\sigma' = |\{c \in \Sigma | c \text{ appears in the sequence}\}|$. The space complexity is $\mathrm{O}\left(N + N^{\frac{2}{3}} \sigma'\right)$ words.

Our main idea is to divide the sequence into $L = \Theta(N^{\alpha})$ subsequences of length which may be zero but not greater than $C = \Theta\left(N^{1-\alpha}\right)$ for some parameter a. Let B_i be the i-th subsequence. We call it a block. For sequences X, Y, we define $X + Y$ as the sequence obtained by concatenating X and Y in this order.

The data structure consists of the following components.

- T_A : A data structure for the sequence A. It can process the following queries.
  - access $A[l : r]$ $(0 \le l \le r < |A|)$ in $\mathrm{O}(T_{1,r-l+1})$ time.
  - insert a character $c\,(\in \Sigma)$ into i-th position of A $(0 \le i \le |A|)$ in $\mathrm{O}(T_2)$ time.
  - delete the i-th character of X $(0 \le i < |A|)$ in $\mathrm{O}(T_3)$ time.
- T_B : A data structure for the array $(|B_0|, |B_1|, \ldots, |B_{L-1}|)$, which is used to compute which block a character in A belongs to. It can process the following queries.
  - increase or decrease the i-th element $(0 \le i < L)$ in $\mathrm{O}(T_4)$ time.
  - calculate $\mathrm{argmin}_i |B_i|$ in $\mathrm{O}(T_5)$ time.
  - calculate $\min\left\{k \middle| \sum_{i=0}^{k} |B_i| \ge a\right\}$ $\left(0 < a \le \sum_i |B_i|\right)$ in $\mathrm{O}(T_6)$ time.
  - insert a value x into i-th position of the array in $\mathrm{O}(T_7)$ time.
  - delete the i-th element of the array in $\mathrm{O}(T_8)$ time.
- $S_{(l,r)}$ $(0 \le l \le r < L)$: A data structure for the ordered set $\{((\text{the multiplicity of } c \text{ in } B_l + \cdots + B_r), c) \mid c \text{ appears in } B_l + \cdots + B_r\}$. It can process the following queries.
  - create an empty set in $\mathrm{O}(T_9)$ time.
  - increment or decrement the multiplicity of character $c\,(\in \Sigma)$ in $\mathrm{O}(T_{10})$ time.
  - compute the multiplicity of a character $c\,(\in \Sigma)$ in $\mathrm{O}(T_{11})$ time.
  - access the largest element in $\mathrm{O}(T_{12})$ time.
  - access the next largest to the last accessed element in $\mathrm{O}(T_{13})$ time.

We introduce new operations moveLeft(i) and moveRight(i). The operation moveLeft(i) moves the first element of i-th block to the $(i-1)$-st block. In such an operation, we only need to modify the following components.

- T_B
- $S_{(0,i-1)}, \ldots, S_{(i-1,i-1)}, S_{(i,i)}, \ldots, S_{(i,L-1)}$

This can be done in $\mathrm{O}(T_4 + LT_{10})$ time. The operation moveRight(i) moves the last element of i-th block to the $(i+1)$-st block. It can be done in the same time in a similar way.

We process the queries by the following method.

delete

Let j be the index of the block that contains the i-th element. It can be computed in $\mathrm{O}(T_6)$ time. T_A and T_B can be modified easily in $\mathrm{O}(T_{1,1} + T_3 + T_4)$ time. We need to modify $S_{(l,r)}$ for all l, r such that $0 \le l \le j \le r < L$. It can be done in $\mathrm{O}\left(L^2 T_{10}\right)$ time.

insert

Let j be $\min\left\{k \middle| \sum_{l=0}^{k} |B_l| \ge i\right\}$. We insert c into the j-th block, and modify the data structure in a similar way to a delete query.

The length of j-th block may become larger than C. In such a case we balance the length of blocks in the following way.

(1) Find a block B_k such that $|B_k| + 1 \le C$.

(2) Operate moveLeft or moveRight several times so that $|B_j|$ decreases by 1, $|B_k|$ increases by 1 and the rest remain.

Step 1. can be done in $\mathrm{O}(T_5)$ time. Step 2. can be done in $\mathrm{O}(L\,(T_4 + LT_{10}))$ time because we call moveLeft or moveRight only $\mathrm{O}(L)$ times.

modes

Let (i, j) be the maximal interval of blocks which is in $A[l:r]$. It can be computed in $\mathrm{O}(T_6)$ time. If $A[l:r]$ does not contain any blocks, $B_i + \cdots + B_j$ stands for an empty sequence and $S_{l,r}$ stands for an empty set below.

It holds that $|A[l:r] \setminus (B_i + \cdots + B_j)| \leq 2C$. It can be said that every mode of $A[l:r]$ is a mode of $B_i + \cdots + B_j$ or appears in $A[l:r] \setminus (B_i + \cdots + B_j)$. If there does not exists such a character c that meets the following conditions

- c is a mode of $A[l:r]$
- c does not appear in $A[l:r] \setminus (B_i + \cdots + B_j)$

then every mode of $A[l:r]$ appears in $A[l:r] \setminus (B_i + \cdots + B_j)$. We scan the elements in $A[l:r] \setminus (B_i + \cdots + B_j)$ and count the occurrences of each character in $A[l:r]$ using $S_{(i,j)}$ and a new ordered set in $\mathrm{O}(T_9 + CT_{10} + T_{1,C} + CT_{11})$ time, and compute the number of occurrences of a mode of $A[l:r]$ in $\mathrm{O}(T_{1,C} + CT_{11} + T_{12})$ time. We can judge if there exists a character satisfying the conditions above using the value and the ordered set. If there does not exist such a character, the enumeration is done. If exists, every mode of $B_i + \cdots + B_j$ is also a mode of $A[l:r]$, so we can enumerate the modes of $A[l:r]$ by the privious scan and the enumeration of the modes of $B_i + \cdots + B_r$, which can be done in $\mathrm{O}(T_{12} + T_{13}\,|output|)$ time. Algorithm 1 denotes the algorithm for the modes query.

In order to keep $L = \Theta(N^\alpha)$ and $C = \Theta(N^{1-\alpha})$, we use the technique for dynamic data structures [9]. We group the blocks into three types p, c, n(previous, current, next). Set the number and size of the blocks as follows.

- $L_\mathrm{p} = \lceil (\frac{N}{2})^\alpha \rceil, C_\mathrm{p} = \lceil (\frac{N}{2})^{1-\alpha} \rceil$
- $L_\mathrm{c} = \lceil N^\alpha \rceil, C_\mathrm{c} = \lceil N^{1-\alpha} \rceil$
- $L_\mathrm{n} = \lceil (2N)^\alpha \rceil, C_\mathrm{n} = \lceil (2N)^{1-\alpha} \rceil$

When we initialize the data structure, all elements are stored in c blocks and initialize the data structure for $L = L_\mathrm{p} + L_\mathrm{c} + L_\mathrm{n}$ blocks.

Algorithm 1: The algorithm for the modes query.

```
Input: range (l, r)
Output: all modes of S[l : r]
Function Main:
    (i, j) ← maximal interval such that
      B_i + ··· + B_j ⊂ A[l : r]
    T ← a new empty ordered set
    for c ∈ A[l : r] \ (B_i + ··· B_j) do
        increment the multiplicity of c in T
    app ← 0
    for (app_c, c) ∈ T do
        app = max(app, app_c +
          (the number of appearences of c in B_i + ··· B_j))
    ans ← ∅
    for (app_c, c) ∈ T do
        if app = app_c +
          (the number of appearences of c in B_i + ··· B_j)
        then
            ans ← ans ∪ {c}
    (app_c, c) ← the top element of S_(i,j)
    while c ≠ NULL and app_c = app do
        ans ← ans ∪ {c}
        (app_c, c) ← the next largest to (app_c, c) in S_(i,j)
    return ans
```

insert

Move elements so that the sum of the elements in n blocks increases by two and that in p blocks decreases by one (unless they are already empty) compared to before the query. To achieve this, we move elements as follows

- insertion into p: p → c, p → c, c → n, c → n
- insertion into c: p → c, c → n, c → n
- insertion into n: p → c, c → n

where x → y means moving the last element of x blocks to y blocks. If all x blocks are empty, it is ignored.

delete

Move elements so that the sum of the elements in n blocks decreases by two (unless they are already empty) and that in p blocks increases by one compared to before the query. To achieve this, we move elements as follow

- insertion into p: c ← n, c ← n, p ← c, p ← c
- insertion into c: c ← n, c ← n, p ← c
- insertion into n: c ← n, p ← c

where x ← y means moving the first element of y blocks to x blocks. If all y blocks are empty, it is ignored.

Lemma 8. *[9] When the length of the string becomes double, all elements are in* n *blocks.*
When the length of the string becomes half, all elements are in p *blocks.*

If the length of the string becomes double, set the blocks as follows

$$(\mathrm{p}, \mathrm{c}, \mathrm{n}) \leftarrow (\mathrm{pp}, \mathrm{p}, \mathrm{c})$$

and if the length of the string becomes half, set the blocks as follows

$$(\mathrm{p}, \mathrm{c}, \mathrm{n}) \leftarrow (\mathrm{c}, \mathrm{n}, \mathrm{nn})$$

where pp (previous to the previous) and nn (next to the next) are other types of blocks and $L_\mathrm{pp}, C_\mathrm{pp}, L_\mathrm{nn}$, and C_nn are defined as follow.

- $L_\mathrm{pp} = \lceil (\frac{N}{4})^\alpha \rceil, C_\mathrm{pp} = \lceil (\frac{N}{4})^{1-\alpha} \rceil$
- $L_\mathrm{nn} = \lceil (4N)^\alpha \rceil, C_\mathrm{nn} = \lceil (4N)^{1-\alpha} \rceil$

We need to add $\mathrm{O}(N^{1+2\alpha})$ extra elements for $S_{(l,r)} (0 \leq l \leq r < L_\mathrm{pp} + L_\mathrm{p} + L_\mathrm{c} + L_\mathrm{n} + L_\mathrm{nn})$ in order to prepare pp blocks and nn blocks and are prepared from when the block reset occured. There are $\Omega(N)$ queries. These operations need
$\mathrm{O}\big(T_5 + T_7 + T_8 + L\,(T_4 + LT_{10}) + N^{2\alpha} T_{10} + \max\big(1, N^{2\alpha-1}\big)\, T_9\big)$
time per query.

Theorem 9. *There exists a data structure for the dynamic range mode enumeration problem in*
$\mathrm{O}\big(T_{1,1} + T_2 + T_3 + T_5 + T_6 + N^\alpha T_4 + N^{2\alpha} T_{10} + \max\big(1, N^{2\alpha-1}\big)\, T_9\big)$

time per insert *and* delete *query and* $\mathrm{O}\big(T_6 + T_{1,\Theta(N^{1-\alpha})} + N^{1-\alpha}T_{11} + T_{12} + T_{13}\,|output|\big)$ *time per* modes *query where* N *is the length of the sequence and* $\sigma = |\Sigma|$.

Proof of Theorem 7. We use balanced binary search trees for T_A and T_B.

We use two balanced binary search trees for each $S_{(l,r)}$. One of them is the one whose key is a character in $B_l + \cdots + B_r$ and value is the number of the occurrences of the character in $B_l + \cdots + B_r$. The other is used as a ordered set $\{(t(c), c) | c \in \Sigma, t(c) > 0\}$, where $t(c)$ is the number of the occurrences of c in $B_l + \cdots + B_r$. Then, following equations hold.

$$
\begin{aligned}
T_{1,a} &= \mathrm{O}(a + \log N)\\
T_2 &= \mathrm{O}(\log N)\\
T_3 &= \mathrm{O}(\log N)\\
T_4 &= \mathrm{O}(\log L) = \mathrm{O}(\log N)\\
T_5 &= \mathrm{O}(\log L) = \mathrm{O}(\log N)\\
T_6 &= \mathrm{O}(\log L) = \mathrm{O}(\log N)\\
T_7 &= \mathrm{O}(\log L) = \mathrm{O}(\log N)\\
T_8 &= \mathrm{O}(\log L) = \mathrm{O}(\log N)\\
T_9 &= \mathrm{O}(1)\\
T_{10} &= \mathrm{O}\big(\log \sigma'\big)\\
T_{11} &= \mathrm{O}\big(\log \sigma'\big)\\
T_{12} &= \mathrm{O}(1)\\
T_{13} &= \mathrm{O}(1)
\end{aligned}
$$

Setting $\alpha = \frac{1}{3}$, we obtain theorem 7. □

3. Concluding Remarks

We introduced a new problem, the dynamic range mode enumeration problem. We found an algorithm for it whose time complexity of a modes query is linear to the output size plus some term. However, the term is larger than the time complexity of a mode query of the dynamic range mode problem. It may be possible to found a new algorithm for the dynamic range mode enumeration problem whose time complexity for a query is equal to that of the dynamic range mode problem except the term depending on the output size.

References

[1] Petersen, H.: Improved Bounds for Range Mode and Range Median Queries, *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM'08, Springer-Verlag, pp. 418–423 (2008).

[2] Chan, T. M., Durocher, S., Skala, M. and Wilkinson, B. T.: Linear-Space Data Structures for Range Minority Query in Arrays, *Algorithm Theory – SWAT 2012* (Fomin, F. V. and Kaski, P., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 295–306 (2012).

[3] Petersen, H. and Grabowski, S.: Range mode and range median queries in constant time and sub-quadratic space, *Inf. Process. Lett.*, Vol. 109, pp. 225–228 (online), DOI: 10.1016/j.ipl.2008.10.007 (2009).

[4] Greve, M., Jørgensen, A. G., Larsen, K. D. and Truelsen, J.: Cell Probe Lower Bounds and Approximations for Range Mode, *Automata, Languages and Programming* (Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F. and Spirakis, P. G., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 605–616 (2010).

[5] Chan, T., Durocher, S., Larsen, K., Morrison, J. and Wilkinson, B.: Linear-Space Data Structures for Range Mode Query in Arrays, *Theory of Computing Systems*, Vol. 55, No. 4, pp. 719–741 (online), DOI: 10.1007/s00224-013-9455-2 (2014).

[6] Sumigawa, K., Chakraborty, S., Sadakane, K. and Satti, S. R.: Enumerating Range Modes, *31st International Symposium on Algorithms and Computation (ISAAC 2020)* (Cao, Y., Cheng, S.-W. and Li, M., eds.), Leibniz International Proceedings in Informatics (LIPIcs), Vol. 181, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 29:1–29:16 (online), DOI: 10.4230/LIPIcs.ISAAC.2020.29 (2020).

[7] El-Zein, H., He, M., Munro, J. I. and Sandlund, B.: Improved Time and Space Bounds for Dynamic Range Mode, *26th Annual European Symposium on Algorithms (ESA 2018)* (Azar, Y., Bast, H. and Herman, G., eds.), Leibniz International Proceedings in Informatics (LIPIcs), Vol. 112, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 25:1–25:13 (online), DOI: 10.4230/LIPIcs.ESA.2018.25 (2018).

[8] Sandlund, B. and Xu, Y.: Faster Dynamic Range Mode, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)* (Czumaj, A., Dawar, A. and Merelli, E., eds.), Leibniz International Proceedings in Informatics (LIPIcs), Vol. 168, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 94:1–94:14 (online), DOI: 10.4230/LIPIcs.ICALP.2020.94 (2020).

[9] Mäkinen, V. and Navarro, G.: Dynamic entropy-compressed sequences and full-text indexes, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 4009 LNCS, pp. 306–317 (2006).