Encodings for Range Minimum Queries over Bounded Alphabets

Seungbum Jo
Chungnam National University, Daejeon, South Korea

Srinivasa Rao Satti
Norwegian University of Science and Technology, Trondheim, Norway

Abstract

Range minimum queries (RMQs) are fundamental operations with widespread applications in database management, text indexing and computational biology. While many space-efficient data structures have been designed for RMQs on arrays with arbitrary elements, there has not been any results developed for the case when the alphabet size is small, which is the case in many practical scenarios where RMQ structures are used. In this paper, we investigate the encoding complexity of RMQs on arrays over bounded alphabet. We consider both one-dimensional (1D) and two-dimensional (2D) arrays. For the 1D case, we present a near-optimal space encoding. For constant-sized alphabets, this also supports the queries in constant time. For the 2D case, we systematically analyze the 1-sided, 2-sided, 3-sided and 4-sided queries and derive lower bounds for encoding space, and also matching upper bounds that support efficient queries in most cases. Our results demonstrate that, even with the bounded alphabet restriction, the space requirements remain close to those for the general alphabet case.

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

Keywords and phrases Range minimum queries, Encoding data structures, Cartesian trees

Digital Object Identifier 10.4230/LIPIcs.CPM.2025.25

Funding *Seungbum Jo*: This work was supported by research fund of Chungnam National University.

1 Introduction

Efficiently processing range queries on arrays has been one of the central problems in computer science with a wide range of applications in areas such as database management, text indexing, computational biology etc. In this paper, we focus on a few different variants of range minimum queries (RMQ) on one- and two-dimensional arrays. Given a query range within an array (that we are allowed to preprocess) a range minimum query returns the position of a smallest element within the query range.

Over the past decades, there has been a significant amount of research on designing space-efficient data structures that support fast queries for the RMQ problem. Almost all of these results focus on arrays with arbitrary elements (and a few on binary arrays), whereas in many real-world scenarios one often encounters input arrays that are not arbitrary but drawn from bounded or small alphabets (but not necessarily binary) - genome sequences for example. Our aim is to investigate the effect of the alphabet size on the size of the RMQ encoding.

In the one-dimensional (1D) case, many classical data structures [1,15,22] have established that RMQ can be answered in constant time using linear space. The space usage is further improved to match the information-theoretic lower bound (up to lower-order terms) [9]. Extending the RMQ from one to two dimensions introduces new complexities. Unlike the 1D case, where the Cartesian tree provides a natural and optimal representation for encoding RMQ answers, the two-dimensional (2D) case lacks a straightforward analogue [6].

In this paper we study *encoding data structures* for RMQ in the 1D and 2D arrays. An encoding data structure for a given set of queries is a structure that stores sufficient information to answer those queries without directly accessing the original input data. On the other hand, an indexing data structure refers to a structure that maintains the original input while maintaining small auxiliary structures to support queries efficiently. Our primary focus is on establishing upper and lower bounds for RMQ encodings when the input array is drawn from a bounded-sized alphabet. Many data structures, such as text indexing structures [7,10,16,17] and rank/select structures on sequences [14,19], can be made compact and fast when the input is drawn from a bounded-sized alphabet rather than an arbitrary one. We explore whether one can exploit the bounded-alphabet input to improve the complexity of 1D and 2D RMQ encodings.

1.1 Previous results on RMQ encodings

1D-RMQ encodings. For the 1D case, any encoding for RMQ requires at least $2n - o(n)$ bits due to its bijective relationship with the Cartesian tree of the input [23]. The best-known result is the $(2n + o(n))$-bit data structure by Fischer and Heun that supports $O(1)$ query time. When the input is highly compressible, one can design data structures whose space usage is parameterized by the compressibility of the Cartesian tree of the input [11, 18] while still supporting queries efficiently. Fischer considered the number of distinct Cartesian trees with bounded alphabets in his Ph.D. thesis (see Section 3.9.1 in [8]) and computed, for each $\sigma \in \{2, 3, 4\}$, a constant c_σ such that storing a Cartesian tree constructed from an input of size n over an alphabet of size σ requires at least $c_\sigma n - \Theta(1)$ bits.

2D-RMQ encodings. When the input is an $m \times n$ 2D array with $m \leq n$, Demaine et al. [6] showed that there is no Cartesian tree-like structure for the 2D case. Specifically, no structure exists that fully encodes the answers to all queries that can be constructed in linear time. They further showed that when $m = n$, any encoding data structure for answering RMQ queries requires $\Omega(n^2 \log n)$ bits. Since the input array can be trivially encoded using $O(n^2 \log n)$ bits (with respect to RMQs), this implies that any encoding data structure uses asymptotically the same space as indexing data structures [3, 24] when $m = \Theta(n)$. Thus, most of the subsequent results focused on the case where $m = o(n)$.

Brodal et al. [3] proposed an $O(nm \cdot \min(m, \log n))$-bit encoding with $O(1)$ query time. Moreover, they proved that any encoding for answering RMQ requires at least $\Omega(nm \log m)$ bits. Brodal et al. [2] proposed an asymptotically optimal $O(nm \log m)$-bit encoding for answering RMQ, although the queries are not supported efficiently using this encoding. For $m = o(n)$, the problem of designing a $o(nm \log n)$-bit data structure for a 2D array that answers queries in sublinear time remains an open problem.

For an $m \times n$ 2D array, Golin et al. [13] considered the following four type of queries:

1. 1-sided: $[1, m] \times [1, j]$ for $1 \leq j \leq n$
2. 2-sided: $[1, i] \times [1, j]$ for $1 \leq i \leq m$, $1 \leq j \leq n$
3. 3-sided: $[1, i] \times [j_1, j_2]$ for $1 \leq i \leq m$, $1 \leq j_1 \leq j_2 \leq n$
4. 4-sided: $[i_1, i_2] \times [j_1, j_2]$ for $1 \leq i_1 \leq i_2 \leq m$, $1 \leq j_1 \leq j_2 \leq n$ (any rectangular range).

Golin et al. [13] provided expected upper bounds and matching lower bounds for the encoding space required to answer RMQ for these four query types assuming the elements of the input array are drawn uniformly at random.

Table 1 Summary of the results of upper and lower bounds for RMQ encodings on $m \times n$ 2D arrays ($m \leq n$) over an alphabet of size σ. The results marked (*) indicate expected space.

Query type	Space (in bits)	Query time	Reference
Upper bounds			
1-sided	$O(\log^2 n)$*		[13]*
	$\min(\sigma, n)\lceil \log m \rceil + \log \binom{n}{\min(\sigma,n)} + o(n)$	$O(1)$	Theorem 5
2-sided	$O(\log^2 n \log m)$*		[13]*
	$O(mn)$	$O(1)$	Theorem 7
	$\sigma \log \binom{n+m+2}{m+1} + o(\sigma n)$	$O(\log \sigma)$	Theorem 10
3-sided	$O(n \log^2 m)$*		[13]*
	$O(mn)$	$O(1)$	Theorem 7
	$n\sigma \log m + O(\sigma n)$	$O(\log \sigma)$	Theorem 13
4-sided	$O(nm)$*		[13]*
	$O(\min(m^2 n, mn \log n))$	$O(1)$	[4]
	$O(mn \log m)$		[2]
	$nm\lceil \log \sigma \rceil + o(mn)$	$O(1)$	Theorem 16, $\sigma = O(1)$
Lower bounds			
1-sided	$\Omega(\log^2 n)$*		[13]*
	$\min(\sigma, n) \log m + \log \binom{n}{\min(\sigma-1,n)}$		Theorem 6
2-sided	$\Omega(\log^2 n \log m)$*		[13]*
	$\Omega(mn)$		Theorem 7
	$\Omega(\sigma m \log \frac{n-4\sigma}{m})$		Theorem 11, $\sigma < n/4 - 4$
3-sided	$\Omega(n \log^2 m)$*		[13]*
	$n \log \binom{m-1}{\sigma-1} = \Omega(n\sigma \log(m/\sigma))$		Theorem 14, $\sigma < m$
4-sided	$\Omega(nm)$*		[13]*
	$\Omega(mn \log m))$		[4]
	$\Omega(mn \log \sigma))$		Theorem 15, $\sqrt{\sigma} \leq \min(m, n/2)$

1.2 Our results

In this paper, we study encoding data structures for RMQ on 1D and 2D arrays over a bounded-sized alphabet. All the encoding results assume a $\Theta(\log n)$-bit word RAM model, where n is the input size. For a 1D array of length n over an alphabet of size σ, we show that any RMQ encoding needs at least $n \log(4 \cos^2(\frac{\pi}{\sigma+2})) - O(\sigma \log n)$ bits (Theorem 3). This shows that even for moderately large alphabet, the lower bound is close to the $2n - O(\log n)$-bit lower bound for the general alphabet. Moreover, we show that for any constant-sized alphabet, one can achieve optimal space usage (up to lower-order terms) while supporting the queries in constant time (Theorem 4).

For a 2D $m \times n$ array with $m \leq n$, we first show (Theorem 5) that $\Theta(n \log m)$ bits are necessary and sufficient to answer 1-sided RMQ (in constant time). We then generalize this bound to $\Theta(\min(n, \sigma) \log m)$ bits when the alphabet size is σ (Theorem 6). For 2-sided and 3-sided RMQ, we show that $\Theta(mn)$ bits are necessary and sufficient for answering queries in constant time (Theorem 7 and Corollary 12). On the other hand, when the input 2D array is over an alphabet of size σ, we show that any 2-sided RMQ encoding requires at least $\sigma \log \binom{n-4\sigma}{m}$ bits (Theorem 11). We also show that one can match this lower bound for some ranges of parameters, by describing a data structure that takes $\sigma \log \binom{n+m+2}{m+1} + o(\sigma n)$ bits which can answer 2-sided RMQ in $O(\log \sigma)$ time (Theorem 10). For 3-sided RMQ, we show that there exists a data structure of size $n\sigma \log m + O(n\sigma)$ bits that answers queries in $O(\log \sigma)$ time (Theorem 13). We also show that the space bound is asymptotically optimal for some range of parameters by showing that any 3-sided RMQ encoding requires at least $n \log \binom{m-1}{\sigma-1}$ bits (Theorem 14). Finally, for the 4-sided RMQ, we first show that $\Theta(mn \log \sigma)$ bits are necessary and sufficient (Theorem 15), and design an encoding that supports 4-sided RMQ in constant time when σ is $O(1)$ (Theorem 16). See Table 1 for a summary of the results on 2D arrays.

The remainder of this paper is organized as follows. In Section 2, we revisit the RMQ problem in the 1D setting over bounded-sized alphabets, establishing tight upper and lower bounds. Section 3 explores the 2D case, where we systematically consider various types of query ranges (namely, 1-, 2-, 3- and 4-sided queries) and derive upper and lower bounds on the size of the encodings, in all cases achieving asymptotically optimal bounds.

We use the following notation throughout the rest of this paper. Given a input 1D (resp. 2D) array and a 1D range $[i, j]$ (resp. a 2D rectangular range $[i_1, j_1] \times [i_2, j_2]$) on the array, $\mathsf{rmq}(i, j)$ (resp. $\mathsf{rmq}(i_1, j_1, i_2, j_2)$) returns the position of the smallest element within the given range. In case of a tie, a tie-breaking rule is applied, always returning the leftmost (resp. top-leftmost) position. Also for a 2D array, (i, j) denotes a position at the i-th row and j-th column, and $[n]$ denotes the set $\{1, \ldots, n\}$.

2 RMQ on 1D array over bounded-sized alphabets

In this section, we consider a data structure for answering RMQ on a 1D array $A[1, n]$ of size n where the array elements are from an alphabet $\Sigma = \{0, \ldots, \sigma - 1\}$ of size σ. We begin with the case when $\sigma = 2$. In this case, RMQ can be answered in $O(1)$ time using an $(n + o(n))$-bit data structure [20] that supports rank_0 and select_0 queries on A in $O(1)$ time. Here, a $\mathsf{rank}_\alpha(i)$ query returns the number of α's in the prefix $A[1, i]$, while a $\mathsf{select}_\alpha(j)$ query returns the position of the j-th occurrence of α in A, for any $\alpha \in \Sigma$ and $i, j \in \{1, \ldots, n\}$. Furthermore, the following lemma shows that this data structure is optimal for answering RMQ, up to additive lower-order terms.

▶ **Lemma 1.** *At least $n - 1$ bits are necessary to answer RMQ on a 1D binary array A of size n.*

Proof. Consider an arbitrary binary array A of length n with $A[n] = 0$. Then for any $i \in \{1, \ldots, n - 1\}$, $A[i] = 0$ if and only if $\mathsf{rmq}(i, n) = i$, since RMQ returns the leftmost position in case of a tie. Consequently, A can be fully reconstructed using only RMQ, which proves the lemma. ◀

For an arbitrary 1D array A of size n, it is known that $2n$ bits are necessary and sufficient (up to lower-order additive terms) to answer RMQ [9]. The lower bound follows from the fact that if two arrays yield different RMQ results, their corresponding Cartesian trees [23] must have distinct shapes. The Cartesian tree of A (denoted $C(A)$) is an unlabeled binary tree defined as follows: (i) The root r of $C(A)$ corresponds to $\mathsf{rmq}(1, n)$. (ii) The left and right subtrees of r are the Cartesian trees of $A[1, \mathsf{rmq}(1, n) - 1]$ and $A[\mathsf{rmq}(1, n) + 1, n]$, respectively, if they exist.

Let's define the *left height* of a binary tree T as the maximum number of left edges on any root to leaf path in the binary tree. From the definition of a Cartesian tree and the tie-breaking rule for RMQ, one can observe that the element corresponding to any node in the Cartesian tree is strictly less the element corresponding to its left child. From this we derive the following observation.

▶ **Proposition 2.** *Given a 1D array A over an alphabet of size σ, the left height of $C(A)$ is at most $\sigma - 1$.*

Genitrini et al. [12] count the number of labeled relaxed binary trees with n internal nodes, where the left height is bounded by σ. Roughly speaking, a relaxed binary tree of size n is a directed acyclic graph consisting of a binary tree with n internal nodes, one leaf, and n pointers. It is constructed from a binary tree of size n, where the first leaf in a reverse

post-order traversal is kept and all other leaves are replaced by pointers. These links may point to any node that has already been visited by the reverse post-order traversal. See Figure 1 for an example. A Cartesian tree with n nodes can be viewed as a relaxed binary tree with n internal nodes by adding dummy left (resp. right) leaves for nodes that have no left (resp. right) children.

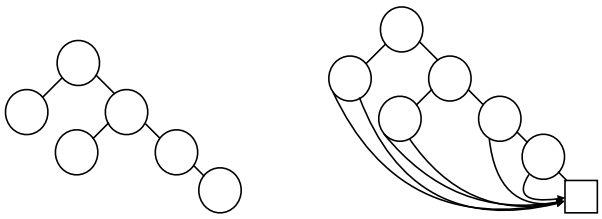


Figure 1 (a) The Cartesian tree of a binary array 1 0 1 0 0 1, and (b) its relaxed binary tree representation. The square node represents a dummy leaf node.

Furthermore, since a Cartesian tree is an unlabeled tree, its relaxed binary tree representation contains a single leaf node. Therefore, for a 1D array A of size n over an alphabet of size σ, the number of distinct Cartesian trees on A is at least the number of distinct relaxed binary trees with n internal nodes and left height at most $\sigma - 1$, since exactly one node is added as a rightmost leaf in its relaxed binary tree. Therefore, from [12], we directly obtain a following theorem.

▶ **Theorem 3** ([12]). *Given a 1D array A of size n over an alphabet of size σ, at least $n \log r_{(\sigma-1)} - O(\sigma \log n)$ bits are necessary to answer RMQ, where $r_\sigma = 4\cos^2(\frac{\pi}{\sigma+3})$.*

For example, when σ is $2, 3, 4, 5$ and 10, at least 1, 1.388, $\log 3 = 1.585$, 1.698 and 1.899 bits per element are required to answer RMQ on A, respectively (see Table 1 in [12] for more examples)[1]. This implies that even for constant-sized alphabets, the space lower bound for answering RMQ remains very close to that of the general case. From an upper bound perspective, when $\sigma = O(1)$, we can obtain an $(n \log r_{\sigma-1} + o(n))$-bit data structure that answers RMQ in $O(1)$ time, as summarized in the following theorem.

▶ **Theorem 4.** *Given a 1D array A of size n over an alphabet of size $\sigma = O(1)$, there exists a data structure of size $n \log r_{\sigma-1} + o(n)$ bits that can answer RMQ in $O(1)$ time.*

Proof. We use a slightly modified version of the data structure from Davoodi et al. [5]. Roughly speaking, their approach decomposes $C(A)$ into $\Theta(n/\ell)$ *micro-trees* of size at most $\ell = (\log n)/4$. Each micro-tree is stored as an index into a precomputed table of $O(2^{2\ell}) = o(n)$ bits, which contains the information of all possible Cartesian trees of size at most ℓ. One can show that these pointers into the precomputed table use $2n + o(n)$ bits. Additionally, the data structure includes $o(n)$-bit auxiliary structures to support the queries in $O(1)$ time.

We modify the data structure such that the precomputed table stores only Cartesian trees of size at most ℓ with left height at most $\sigma - 1$. As a result, one can show that the total space required to store the pointers for all micro-trees is reduced to $n \log r_{\sigma-1} + o(n)$ bits, while all other auxiliary structures remain unchanged. ◀

[1] for $\sigma \in \{2, 3, 4\}$, Fischer [8] obtained the same result.

3 RMQ on 2D array over bounded-sized alphabets

In this section, we consider a data structure for answering RMQ on a 2D array $A[1,m][1,n]$ of size $N = mn$ over an alphabet $\Sigma = \{0,\ldots,\sigma-1\}$ of size σ with $m \leq n$.

3.1 1-sided queries

Recall that for 1-sided RMQ, the query range is restricted to $[1,m]\times[1,i]$ for $i \in [n]$. We begin by considering the upper and lower bounds of the data structure for the general case.

▶ **Theorem 5.** *For an $m \times n$ array, (1) at least $n\log m$ bits are necessary to answer 1-sided RMQ, and (2) there exists an $(n\log m + O(n))$-bit data structure that answers 1-sided RMQ in $O(1)$ time.*

Proof. (1) Consider a set $\mathcal{B}$ containing all possible $m \times n$ arrays where each array has exactly one entry with value i in the i-th column, while all other entries are set to $n+1$. The total number of such arrays is $|\mathcal{B}| = m^n$. Then, any array $B \in \mathcal{B}$ can be reconstructed using 1-sided RMQ on B, since the position of the value i in column i is given by $\mathsf{rmq}(1,m,1,i)$. Thus, at least $n\log m$ bits are necessary to answer 1-sided RMQ.
(2) For all indices j such that $j = 1$ or $\mathsf{rmq}(1,m,1,j-1) \neq \mathsf{rmq}(1,m,1,j)$, we store the row positions of $\mathsf{rmq}(1,m,1,j)$ using at most $n\lceil\log m\rceil$ bits (in this case, $\mathsf{rmq}(1,m,1,j)$ is in the j-th column). Additionally, we maintain a bit string B of size n to mark all such indices j, along with an auxiliary structure of $o(n)$ bits that supports rank and select queries on it in $O(1)$ time [20]. Thus, the total size of the data structure is at most $n\log m + O(n)$ bits. To answer $\mathsf{rmq}(1,m,1,i)$ in $O(1)$ time, we proceed as follows: (i) Identify the rightmost $j \leq i$ such that $B[j] = 1$ using rank and select queries on B, and (ii) return (r,j), where r is the stored row position (of $\mathsf{rmq}(1,m,1,j)$) corresponding to the j-th column. ◀

When A is defined over an alphabet of size σ, there exists at most $\min(\sigma,n)$ indices j where $\mathsf{rmq}(1,m,1,j-1) \neq \mathsf{rmq}(1,m,1,j)$. Therefore, the data structure of the Theorem 5 takes at most $\min(\sigma,n)\lceil\log m\rceil + \log\binom{n}{\min(\sigma,n)} + o(n)$ bits [20], which is smaller than $n\log m + O(m)$ bits when $\sigma = o(n)$. Furthermore, the following theorem shows that this is optimal when $\sigma \leq n$, up to additive lower-order terms (if $\sigma > n$, the data structure of Theorem 5 gives a data structure with optimal space usage).

▶ **Theorem 6.** *For an $m \times n$ array defined over an alphabet of size $\sigma \leq n$, at least $(\sigma-1)\log m + \log\binom{n}{\sigma-1}$ bits are necessary to answer 1-sided RMQ.*

Proof. We use a similar argument as in the proof of (1) in Theorem 5. Let $n \geq j_0 > j_1 > \cdots > j_{\sigma-2} \geq 1$ be $\sigma-1$ distinct column positions. Define $\mathcal{B}$ as the set of all possible $m \times n$ arrays where: (i) For each $k \in \{0,\ldots,\sigma-2\}$, exactly one entry in the j_k-th column has the value k. (ii) All other entries are set to $\sigma-1$. Since $j_0,\ldots,j_{\sigma-1}$ can be chosen arbitrarily within the given range, the total size of $\mathcal{B}$ is $m^{\sigma-1}\binom{n}{\sigma-1}$. Furthermore, any array $B \in \mathcal{B}$ can be reconstructed using 1-sided RMQ on B. Specifically, for each $k \in \{0,\ldots,\sigma-2\}$, the position of the value k is given by $\mathsf{rmq}(1,m,1,j'_k)$, where j'_k is the k-th rightmost column such that the column position of $\mathsf{rmq}(1,m,1,j'_k)$ is j'_k. ◀

3.2 2-sided queries

In this section, we consider the case where the query range is restricted to $[1,i]\times[1,j]$ with $i \in [m]$ and $j \in [n]$. As with 1-sided queries, we first give the upper and lower bounds of the data structure for the general case.

▶ **Theorem 7.** *For an $m \times n$ array, (1) at least $\Omega(mn)$ bits are necessary to answer 2-sided RMQ, and (2) there exists an $O(mn)$-bit data structure that answers 2-sided RMQ in $O(1)$ time.*

Proof. (1) Let $\mathcal{B}_i$ be the set of all possible 1D arrays of size n such that for each $B_i \in \mathcal{B}_i$: (i) $B_i[1] = n \cdot i - 1$, and (ii) for $j \in [n] \setminus \{1\}$, $B_i[j]$ is either $B_i[j-1]$ or $B_i[j-1] - 1$. The size of $\mathcal{B}_i$ is 2^{n-1}. Furthermore, any array $B_i \in \mathcal{B}_i$ can be reconstructed using $\mathsf{rmq}(1, j)$ for all $j \in [n]$, based on the tie-breaking rule for RMQ.

Next, let $\mathcal{B}$ be the set of all possible $m \times n$ arrays where the i-th row is an array from $\mathcal{B}_{m-i}$. The size of $\mathcal{B}$ is $2^{m(n-1)}$. Moreover, any array $B \in \mathcal{B}$ can be reconstructed using 2-sided RMQ on B, since the i-th row of B can be determined from $\mathsf{rmq}(1, i, 1, j)$ for all $j \in [n]$ (note that $\mathsf{rmq}(1, i, 1, j)$ always lies in the i-th row by construction - since any element in the first $(i-1)$ rows is larger than every element in the i-th row). Thus, at least $\log|\mathcal{B}| = \Omega(mn)$ bits are necessary to answer 2-sided RMQ.

(2) Given an input array A, define m distinct 1D arrays $A_1, \ldots, A_m$ of size n where for each $i \in [m]$ and $j \in [n]$, $A_i[j] = \min_{k \in [i]} A[k][j]$. We then construct a data structure to answer RMQ on these arrays in $O(1)$ time on each of these m arrays (we do not store the arrays), using a total of $O(mn)$ bits [9]. Additionally, we maintain n data structures for answering RMQ on each column of A in $O(1)$ time, using another $O(mn)$ bits. To answer the 2-sided RMQ query $\mathsf{rmq}(1, i, 1, j) = (i', j')$, we proceed as follows: (i) Compute j' in $O(1)$ time by $\mathsf{rmq}(1, j)$ on A_i, and (ii) compute i' in $O(1)$ time by $\mathsf{rmq}(1, i)$ on the j'-th column of A. ◀

Next, consider the case where the input array A is defined over an alphabet $\Sigma = \{0, \ldots, \sigma - 1\}$ of size σ. For each position (i, j), we refer to the range $[1, i] \times [1, j]$ as the 2-sided region defined by (i, j). Let P be a set of all positions in A that serve as answers to some 2-sided RMQ on A. For any two positions (i, j) and (i', j') of A, we say that (i', j') *dominates* (i, j) if and only if $i \leq i'$ and $j \leq j'$. Because of the tie breaking rule, it follows that for any two distinct positions (i, j) and (i', j') in P if (i', j') dominates (i, j), then $A[i, j] > A[i', j']$. For example in a 2D array in Figure 2(a), two positions $(1, 2)$ and $(2, 3)$ are in P, and since the position $(2, 3)$ dominates the position $(1, 2)$, $A[1, 2] = 2$ is greater than $A[2, 3] = 0$. We partition P into set of *staircases* $S_0, S_2, \ldots, S_{\sigma-1}$ such that for any ℓ, S_ℓ is a set of all positions (i, j) with $A[i, j] = \ell$. Then from the definition, we can directly observe the following:

▶ **Proposition 8.** *Consider staircases $S_1, S_2, \ldots,$ within an $m \times n$ array defined over an alphabet of size σ, the following holds for any ℓ:*

- **(a)** *No position in S_ℓ is dominated by any other position within the same staircase.*
- **(b)** *The i-th bottom-most position in S_ℓ coincides with the i-th leftmost position in S_ℓ.*
- **(c)** *The union of 2-sided regions defined by the positions in S_ℓ contains no value smaller than ℓ.*

For each S_ℓ, we define a lattice path L_ℓ from $(m + 1, 0)$ to $(0, n + 1)$ that satisfies the following conditions: (i) L_ℓ moves only north or east and passes through all positions in S_ℓ as its turning points in a clockwise order, and (ii) for any position (i, j) in the array, there exists a position in S_ℓ dominated by (i, j) if and only if (i, j) lies on or below L_ℓ. Moreover, L_ℓ can be efficiently encoded from the following lemma:

▶ **Lemma 9** ([3])**.** *L_ℓ can be encoded as a bitstring B_ℓ of size $m + n + 2$ containing $m + 1$ ones, where $B_\ell[p]$ is 0 if and only if the p-th move of L_ℓ is north. Using $O(1)$ select queries on this bitstring, one can whether a position (i, j) lies on, below, or above L_ℓ.*

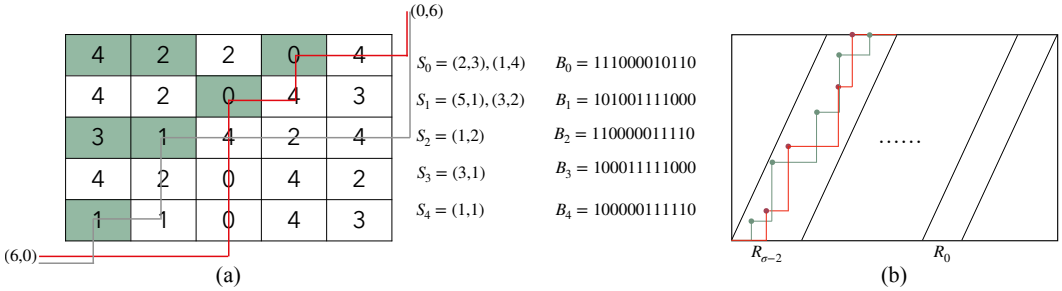


Figure 2 (a) An $m \times n$ array over an alphabet of size 5, where the green positions represent the set P, partitioned into staircases $S_0, \ldots, S_4$. The red and gray lines represent the lattice path L_0 and L_1, respectively. (b) Two distinct lattice paths in the region $R_{\sigma-2}$ and their corresponding staircases (denoted by circular points).

We can maintain σ bitstrings $B_0, \ldots, B_\ell$ of Lemma 9 using $\sigma \log \binom{n+m+2}{m+1} + o(\sigma n)$ bits [20] that support rank and select queries in $O(1)$ time. Given any position (i, j) and $\ell \in \Sigma$, we can determine whether the value at $\mathsf{rmq}(1, i, 1, j)$ is at most ℓ in $O(1)$ time by Lemma 9 and Proposition 8. Consequently, the 2-sided RMQ $\mathsf{rmq}(1, i, 1, j)$ can be answered in $O(\log \sigma)$ time performing a binary search to find the smallest ℓ where (i, j) lies on or below L_ℓ After finding such ℓ, we report $(m - i' + 1, j')$ as the answer, where i' and j' denote the number of zeros and ones in B_ℓ up to the j-th 1, respectively (i.e., $(m - i' + 1, j')$ is the position in S_ℓ dominated by (i, j)). Both i' and j' can be found in $O(1)$ time using rank and select queries on B_ℓ. We summarize the result in the following theorem.

▶ **Theorem 10.** *For an $m \times n$ array A defined over an alphabet of size σ, there exist a data structures using $\sigma \log \binom{n+m+2}{m+1} + o(\sigma n)$ bits that answers 2-sided RMQ in $O(\log \sigma)$ time.*

Note that for any $\sigma = o(m/\log m)$, the data structure of Theorem 10 uses asymptotically less space than the data structure of Theorem 7. Finally, we consider the space lower bound for answering 2-sided RMQ on a 2D array with a bounded alphabet. The following theorem implies that when $\sigma \leq n/c$ for any constant $c > 4$, the data structure in Theorem 10 achieves asymptotically optimal space usage.

▶ **Theorem 11.** *For an $m \times n$ array defined over an alphabet of size $\sigma < n/4 - 4$, at least $\sigma \log \binom{n-4\sigma}{m}$ bits are necessary to answer 2-sided RMQ.*

Proof. For an $m \times n$ array and $n' = n - 4\sigma$, we define $\sigma - 1$ parallelogram regions $R_{\sigma-2}, R_{\sigma-3}, \ldots, R_0$ from the leftmost part of the array, where each region R_ℓ is determined by the four positions: $(1, n' + 4(\sigma - 2 - \ell - 1) + 1)$, $(1, n' + 4(\sigma - 2 - \ell))$, $(m, 4(\sigma - 2 - \ell - 1) + 1)$, and $(m, 4(\sigma - 2 - \ell))$. For each region R_ℓ, we construct a staircase S_ℓ, which consists of the positions in R_ℓ whose values are ℓ. All other positions in the array are assigned the value $\sigma - 1$. Let $\mathcal{A}$ be the set of all possible such arrays.

Now, consider two distinct arrays A_1 and A_2 in $\mathcal{A}$ where the staircases S_ℓ differ. Denote them as S^1_ℓ and S^2_ℓ, respectively. Without loss of generality, let (i_1, j_1) and (i_2, j_2) be the leftmost positions in $S^1_\ell \setminus S^2_\ell$ and $S^2_\ell \setminus S^1_\ell$, respectively, with $i_1 < i_2$. In this case, (i_1, j_1) cannot dominate any positions in S^2_ℓ, implying that the answers to the query $\mathsf{rmq}(1, i_1, 1, j_1)$ on A_1 and A_2 are different. Consequently, at least $\log |\mathcal{A}|$ bits are necessary to answer the 2-sided RMQ on an $m \times n$ array.

To derive a lower bound on the size of $\mathcal{A}$, observe that for each region R_ℓ, we can define a lattice path L_ℓ that does not cross the region boundaries. This path starts at the bottom-left corner of R_ℓ, moves only north or east, and ends at the top-right corner of R_ℓ. Since

each distinct lattice path corresponds to a distinct staircase, defined by the positions of its turning points (in a clockwise order), we obtain at least $\binom{n-4\sigma}{m}$ possible lattice paths for each region R_ℓ [21]. See Figure 2(b) for an example. This implies that $\log |\mathcal{A}|$ is at least $\log \binom{n-4\sigma}{m}^{\sigma} = \Omega(\sigma m \log \frac{n-4\sigma}{m})$. ◀

3.3 3-sided queries

In this section, we consider the case where the query range is restricted to $[1, i] \times [j_1, j_2]$ with $i \in [m]$ and $1 \leq j_1 \leq j_2 \leq n$. Since a 2-sided query is a special case of a 3-sided query with $j_1 = 1$, Theorem 7 implies that at least $\Omega(mn)$ bits are necessary to answer 3-sided queries. Furthermore, the following corollary shows that this space lower bound is also asymptotically tight for 3-sided queries.

▶ **Corollary 12.** *For an $m \times n$ array A, there exists an $O(mn)$-bit data structure that answers 3-sided RMQ in $O(1)$ time.*

Proof. We construct the same data structure as in the proof of Theorem 7 (2) using $O(mn)$ bits. To answer the 3-sided RMQ query $\mathsf{rmq}(1, i, j_1, j_2) = (i', j')$ in $O(1)$ time, we first compute j' by $\mathsf{rmq}(j_1, j_2)$ on A_i, and compute i' by $\mathsf{rmq}(1, i)$ on the j'-th column of A. ◀

Next, consider the case where the input array A is defined over an alphabet $\Sigma = \{0, \ldots, \sigma - 1\}$ of size σ. For each $k \in \Sigma$, define a 1D array C_k of size n such that $C_k[j] = i$ where i is the smallest row index such that (i) $A[i][j] = k$ and (ii) all preceding values in the column, $A[1][j], \ldots, A[i-1][j]$, are greater than k. If j-th column does not have the value k, set $C_k[j] = m + 1$, and if j-th column has the value k, but does not satisfy the condition (ii), set $C_k[j] = 0$. We maintain the arrays $C_0, \ldots, C_{\sigma-1}$ along with RMQ data structures that allow queries to be answered in $O(1)$ time, using a total of $n\sigma \log m + O(\sigma n)$ bits [9].

Given a 3-sided range $[1, i] \times [j_1, j_2]$ on A and a value $k \in \Sigma$, let j_k be the result of $\mathsf{rmq}(j_1, j_2)$ on C_k. Then the value k exists in the range if and only if $C_k[j_k] \leq i$. Thus, we can determine whether k exists in the range in $O(1)$ time and compute the 3-sided RMQ in $O(\log \sigma)$ time by performing a binary search to find the smallest k that exists in the range. We summarize this result in the following theorem.

▶ **Theorem 13.** *For an $m \times n$ array defined over an alphabet of size σ, there exists a data structure of size $n\sigma \log m + O(\sigma n)$ bits that supports 3-sided RMQ in $O(\log \sigma)$ time.*

Compared to the general case, the above data structure requires less space when $\sigma = o(m/\log m)$, and when $\sigma = O(1)$, it answers 3-sided RMQ in $O(1)$ time.

In the general case, the optimal space data structures for answering 2-sided and 3-sided RMQ require asymptotically the same space by Theorem 7 and Corollary 12. However, when the alphabet size is bounded by σ, the data structure from Theorem 10 uses asymptotically less space than the data structure of Theorem 13 when $m = o(n)$. Finally, we consider the space lower bound for answering 3-sided RMQ on a 2D array with a bounded alphabet. The following theorem implies that when $\sigma = o(m)$, the data structure in Theorem 13 achieves asymptotically optimal space usage.

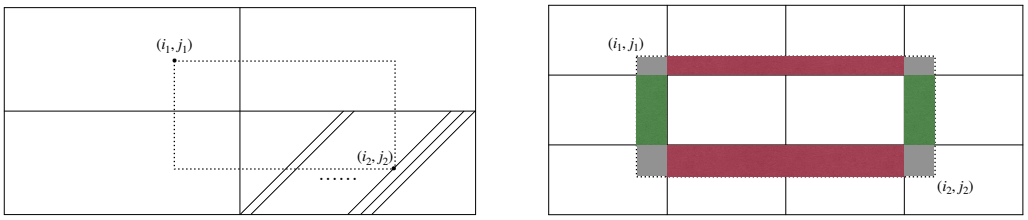
▶ **Theorem 14.** *For an $m \times n$ array defined over an alphabet of size $\sigma < m$, at least $n \log \binom{m-1}{\sigma-1} = \Omega(n\sigma \log(m/\sigma))$ bits are necessary to answer 3-sided RMQ.*

Proof. (1) Let $\mathcal{C}$ be the set of all possible 1D arrays of size m such that for each $C \in \mathcal{C}$: (i) for $k \in \{0, \ldots, \sigma - 2\}$, C has exactly one occurrence of k at the position i_k, (ii) $i_0 > i_1 > \cdots > i_{\sigma-2} > 1$, and (iii) all other positions have the value $\sigma - 1$. Then the size of $\mathcal{C}$ is $\binom{m-1}{\sigma-1}$. Next,

let $\mathcal{B}$ be the set of all possible $m \times n$ arrays where each column is chosen from $\mathcal{C}$. The size of B is $\binom{m-1}{\sigma-1}^n$. Moreover, any array $B \in \mathcal{B}$ can be reconstructed using 3-sided RMQ on B since j-th column of B can be reconstructed from $\mathsf{rmq}(1, i, j, j)$ for all $i \in [n]$, which proves the theorem. Specifically, for any $k \in \{0, \ldots, \sigma - 2\}$, the value k appears at $B[i_k][j]$ where i_k is the index of the k-th row from the bottom that satisfies $\mathsf{rmq}(1, i, j, j) \neq \mathsf{rmq}(1, i-1, j, j)$. ◀

3.4 4-sided queries

In this section, we consider the case where the query range is an arbitrary rectangular region. When the alphabet size is unbounded, there exists a data structure using $O(\min(m^2 n, mn \log n))$ bits that answers RMQ in $O(1)$ time, while at least $\Omega(mn \log m)$ bits are necessary for answering RMQ on A [3]. A $\Theta(mn \log m)$-bit encoding is also known [2], although it does not support the queries efficiently. Here, we focus on the case the case where the input array A is defined over a bounded alphabet $\Sigma = \{0, \ldots, \sigma - 1\}$ of size σ. First, we give a lower bound on the space required for the data structure, as stated in the following theorem.



(a) (b)

Figure 3 (a) A single block of size $\sqrt{\sigma} \times 2\sqrt{\sigma}$ of an $m \times n$ array in the set $\mathcal{A}$ used in the proof of Theorem 15, (b) to answer RMQ in the rectangular range (represented by the dotted line): (i) the candidate answer in the gray range can be found using RMQ on individual blocks, (ii) in the red range using RMQ on A_r, (iii) in the green range using RMQ on A_c, and (iv) in the white range using RMQ on A_{rc}.

▶ **Theorem 15.** *For an $m \times n$ array defined over an alphabet $\{0, \ldots, \sigma\}$ of size $\sigma + 1$, at least $\Omega(mn \log \sigma)$ bits are necessary to answer 4-sided RMQ, assuming $\sqrt{\sigma} \leq \min(m, n/2)$.*

Proof. Our proof is analogous to the $\Omega(mn \log m)$-bit lower bound proof of Brodal et al. [3]. First, divide an input array into blocks of size $\sqrt{\sigma} \times 2\sqrt{\sigma}$, and further divide each block into four sub-blocks of size $\frac{\sqrt{\sigma}}{2} \times \sqrt{\sigma}$. Next, assign values to each block using the following procedure:

- Assign the odd values from $\{0, \ldots, \sigma - 1\}$ to the upper-left sub-block in increasing order, following a row-major order.
- Assign the even values from $\{0, \ldots, \sigma - 1\}$ to the $\frac{\sqrt{\sigma}}{2}$ anti-diagonals in the bottom-right sub-block, such that the values in each anti-diagonal are larger than those in the anti-diagonals to the right.
- Assign σ to all the remaining positions in the block.

Let $\mathcal{A}$ be the set of all $m \times n$ arrays where values are assigned according to the above procedure. Since, for any array $A \in \mathcal{A}$, each anti-diagonal within any sub-block of A forms a permutation, and there are $\frac{mn}{2\sigma}$ blocks in A, the total size of $\mathcal{A}$ is at least $(\sqrt{\sigma}/2)!^{\frac{\sqrt{\sigma}}{2} \cdot \frac{mn}{2\sigma}} = (\sqrt{\sigma}/2)!^{\frac{mn}{4\sqrt{\sigma}}}$. Thus, $\log |\mathcal{A}|$ is $\Omega(mn \log \sigma)$. Now, consider two distinct arrays A_1 and A_2 in $\mathcal{A}$ where $A_1[i_2, j_2] < A_2[i_2, j_2]$. From the assignment procedure, the position (i_2, j_2) lies on

an anti-diagonal in the bottom-right sub-block of some block (note that $A_1[i,j] = A_2[i,j]$ for every position (i,j) in the other three sub-blocks of every block). There exists another position (i_1, j_1) in the upper-left sub-block of the same block such that $A_1[i_2,j_2] < A_2[i_1,j_1] < A_2[i_2,j_2]$ (see Figure 3(a) for an example). Thus, $\mathsf{rmq}(i_1,j_1,i_2,j_2)$ on A_1 returns (i_2,j_2), while the same query on A_2 returns (i_1,j_1), This implies that at least $|\mathcal{A}|$ different $m \times n$ arrays have distinct RMQ answers. ◀

Theorem 15 implies that if $\sqrt{\sigma} \leq \min(m, n/2)$, an $(nm\lceil\log\sigma\rceil + O(mn))$-bit indexing data structure (i.e., a data structure that explicitly stores the input array) of Brodal et al. [3], which answers RMQ in $O(1)$ time, uses asymptotically optimal space for the bounded-alphabet case. This means that in most scenarios, restricting the alphabet size provides little advantage in terms of space efficiency compared to the general case. However, when $\sigma = O(1)$, the $O(mn)$ term in the above indexing data structure of [3] can dominate the space for storing the input. In the following theorem, we show that this can be improved to $o(mn)$ bits while still supporting $O(1)$ query time[2].

▶ **Theorem 16.** *For an $m \times n$ array A defined over an alphabet of size $\sigma = O(1)$, there exists a data structure of size $mn\lceil\log\sigma\rceil + o(mn)$ bits that supports 4-sided RMQ in $O(1)$ time.*

Proof. Let $c = \frac{1}{2}\sqrt{\log_\sigma mn}$. Then we partition A into blocks of size $c \times c$ and construct the following structures:

- To efficiently handle queries whose range falls entirely within a single block, we store an index data structure from Brodal et al. [3] for each block as follows. We first store a precomputed table of size $2^{O(c^2\log\sigma)} = O((mn)^{1/4})$, which contains all possible index structures from [3] for blocks of size $c \times c$. Since each block can be represented as an index into this precomputed table using $\frac{1}{4}\log mn$ bits, the total space required for storing indices of all blocks is $\frac{mn}{4c^2} \cdot \log mn = mn\lceil\log\sigma\rceil$ bits. As we maintain these index structures, we can also access any position in the array in $O(1)$ time.
- Next, we define an auxiliary $m \times n/c$ array A_r, where each entry is given by $A_r[i][j] = A[i][j']$, where j' is a column position of $\mathsf{rmq}(i,i,(j-1)c+1,jc)$ on A. We maintain the index data structure of [3] on A_r, using $O(\frac{mn\log\sigma}{c}) = o(mn)$ bits. This allows us to access any position in A_r in $O(1)$ time and answer RMQ on A in $O(1)$ time when both the leftmost and rightmost columns of the query range align with block boundaries. Note that the exact column position of the answer can be determined using the RMQ structure on individual blocks.
  Similarly, we define an auxiliary $m/c \times n$ array A_c and maintain the same structure using $o(mn)$ bits so that we can answer RMQ on A in $O(1)$ time when both the topmost and bottommost rows in the query range align with block boundaries.
- Finally, let A_{rc} be an $m/c \times n/c$ array, where each entry is given by $A_{rc}[i][j] = A[i'][j']$ where (i',j') is $\mathsf{rmq}((i-1)c+1, ic, (j-1)c+1, jc)$ on A. We maintain the index data structure of [3] on A_{rc} using $O(\frac{mn\log\sigma}{c^2}) = o(mn)$ bits. This allows us to access any position in A_{rc} in $O(1)$ time and answer RMQ on A in $O(1)$ time when all boundaries of the query range align with block boundaries.

Any rectangular query range on A can be partitioned into the following regions: (i) at most four rectangular regions fully contained within a single block, (ii) at most two rectangular regions where the leftmost and rightmost columns align with block boundaries, (iii) at

[2] In fact, the result of [3] provides a trade-offs between index size and query time - if only $o(mn)$-bit space is used for the index, the query time increases to $\omega(1)$.

most two rectangular regions where the topmost and bottommost rows align with block boundaries, and (iv) at most one rectangular region where all four boundaries align with block boundaries. Using the structures described above, we can determine the position and value of the minimum element in each partitioned region in $O(1)$ time. See Figure 3(b) for an example. The final result is obtained in $O(1)$ time by returning the position of the minimum element among them. ◀

4 Conclusions

In this paper, we studied encoding data structures for RMQ on 1D and 2D arrays when the alphabet size is bounded. Most of our data structures use asymptotically optimal space across a wide range of alphabet sizes. We also compare our results to the general case with no restrictions on the alphabet size. For 1D arrays, the space usage differs only slightly, even for a constant-sized alphabet. For 2D arrays, the data structure asymptotically requires less space compared to the general case over a wider range of alphabet sizes, from both upper and lower bound perspectives. For example, for an $n \times m$ 2D array with $m \leq n$, we showed that the space usage can be improved for all types of queries when the alphabet size $\sigma = o(m/\log m)$.

It would be interesting to explore other parameters, such as compressed matrix size, that could further improve space usage for encoding data structures for RMQ.

References

1 Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/J.TCS.2003.05.002.

2 Gerth Stølting Brodal, Andrej Brodnik, and Pooya Davoodi. The encoding complexity of two dimensional range minimum data structures. In *ESA*, volume 8125 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2013. doi:10.1007/978-3-642-40450-4_20.

3 Gerth Stølting Brodal, Pooya Davoodi, Moshe Lewenstein, Rajeev Raman, and Srinivasa Rao Satti. Two dimensional range minimum queries and Fibonacci lattices. *Theor. Comput. Sci.*, 638:33–43, 2016. doi:10.1016/J.TCS.2016.02.016.

4 Gerth Stølting Brodal, Pooya Davoodi, and Srinivasa Rao Satti. On space efficient two dimensional range minimum data structures. *Algorithmica*, 63(4):815–830, 2012. doi:10.1007/S00453-011-9499-0.

5 Pooya Davoodi, Rajeev Raman, and Srinivasa Rao Satti. Succinct representations of binary trees for range minimum queries. In *Computing and Combinatorics - 18th Annual International Conference, COCOON 2012, Sydney, Australia, August 20-22, 2012. Proceedings*, volume 7434 of *Lecture Notes in Computer Science*, pages 396–407. Springer, 2012. doi:10.1007/978-3-642-32241-9_34.

6 Erik D. Demaine, Gad M. Landau, and Oren Weimann. On Cartesian trees and range minimum queries. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming*, pages 341–353, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-02927-1_29.

7 Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. An alphabet-friendly FM-index. In *SPIRE*, volume 3246 of *Lecture Notes in Computer Science*, pages 150–160. Springer, 2004. doi:10.1007/978-3-540-30213-1_23.

8 Johannes Fischer. *Data structures for efficient string algorithms.* PhD thesis, Ludwig-Maximilians-Universität München, 2007.

9 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM J. Comput.*, 40(2):465–492, 2011. doi:10.1137/090779759.

10 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020. doi:10.1145/3375890.

11 Pawel Gawrychowski, Seungbum Jo, Shay Mozes, and Oren Weimann. Compressed range minimum queries. *Theor. Comput. Sci.*, 812:39–48, 2020. doi:10.1016/J.TCS.2019.07.002.

12 Antoine Genitrini, Bernhard Gittenberger, Manuel Kauers, and Michael Wallner. Asymptotic enumeration of compacted binary trees of bounded right height. *J. Comb. Theory A*, 172:105177, 2020. doi:10.1016/J.JCTA.2019.105177.

13 Mordecai J. Golin, John Iacono, Danny Krizanc, Rajeev Raman, Srinivasa Rao Satti, and Sunil M. Shende. Encoding 2D range maximum queries. *Theor. Comput. Sci.*, 609:316–327, 2016. doi:10.1016/J.TCS.2015.10.012.

14 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644250.

15 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.

16 Gregory Kucherov and Yakov Nekrich. Full-fledged real-time indexing for constant size alphabets. *Algorithmica*, 79(2):387–400, 2017. doi:10.1007/S00453-016-0199-7.

17 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Text indexing and searching in sublinear time. In *CPM*, volume 161 of *LIPIcs*, pages 24:1–24:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CPM.2020.24.

18 J. Ian Munro, Patrick K. Nicholson, Louisa Seelbach Benkner, and Sebastian Wild. Hypersuccinct trees - new universal tree source codes for optimal compressed tree data structures and range minima. In *ESA*, volume 204 of *LIPIcs*, pages 70:1–70:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.70.

19 Gonzalo Navarro. Wavelet trees for all. *J. Discrete Algorithms*, 25:2–20, 2014. doi:10.1016/J.JDA.2013.07.004.

20 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43, 2007. doi:10.1145/1290672.1290680.

21 Masako Sato. Note on the number of minimal lattice paths restricted by two parallel lines. *Discret. Math.*, 44(1):117–121, 1983.

22 Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988. doi:10.1137/0217079.

23 Jean Vuillemin. A unifying look at data structures. *Commun. ACM*, 23(4):229–239, 1980. doi:10.1145/358841.358852.

24 Hao Yuan and Mikhail J. Atallah. Data structures for range minimum queries in multidimensional arrays. In *SODA*, pages 150–160. SIAM, 2010. doi:10.1137/1.9781611973075.14.