

Improved Range Minimum Queries

Héctor Ferrada and Gonzalo Navarro

Center of Biotechnology and Bioengineering,
Department of Computer Science, University of Chile.
Beauchef 851, Santiago, Chile
`{hferrada|gnavarro}@dcc.uchile.cl`

Abstract

Fischer and Heun [SICOMP 2011] proposed the first Range Minimum Query (RMQ) data structure on an array $A[1, n]$ that uses $2n + o(n)$ bits and answers queries in $O(1)$ time without accessing A . Their scheme converts the Cartesian tree of A into a general tree, which is represented using DFUDS. We show that, by using instead the BP representation, the formula becomes simpler since border conditions are eliminated. This leads to the fastest and most compact practical implementation to date.

Introduction

The Range Minimum Query (RMQ) problem is, given an array $A[1, n]$ with elements from a totally ordered set, build a data structure that receives any pair of positions $1 \leq i \leq j \leq n$ and returns

$$\text{rmq}_A(i, j) = \operatorname{argmin}_{i \leq k \leq j} A[k],$$

that is, the position of a minimum value in $A[i, j]$. In many cases one prefers the leftmost position when there are ties.

The RMQ problem is a fundamental one and has a long history, intimately related to another key problem: the LCA (lowest common ancestor) problem on general ordinal trees is, given nodes u and v , return $\text{lca}(u, v)$, the lowest node that is an ancestor of both u and v . Gabow et al. [1] showed that RMQs can be reduced to computing LCAs on a particular tree, called the *Cartesian tree* [2] of $A[1, n]$. Later, Berkman and Vishkin [3] showed that the LCA problem on any tree can be reduced to an RMQ problem, on an array derived from the tree. In this array, consecutive entries differ by ± 1 . Bender and Farach [4] then gave a solution for this so-called ± 1 -RMQ problem in constant time and linear space (i.e., $O(n)$ words). Sadakane [5] improved the space of that solution, showing that LCAs on a tree of n nodes can be handled in constant time using $2n + o(n)$ bits (including the tree representation [6]). Finally, Fischer and Heun [7] showed that the Cartesian tree can be represented using $2n + o(n)$ bits so that RMQs on A can be transformed into LCA queries on the succinct tree, and this lead to an RMQ solution that also uses $2n + o(n)$ bits and *does not need to access A at query time*.

Fischer and Heun's solution has become a fundamental building block for many succinct data structures, for example for ordinal trees [5, 8, 9], suffix trees [5, 10], document retrieval [11, 12], two-dimensional grids [13], Lempel-Ziv parsing [14], etc.

Their RMQ computation [7] uses three kinds of operations: several `rank/selects` on bitvectors [15, 16], one ± 1 -RMQ [4], and one `open` on parentheses [6]. Although all can be implemented in constant time, in practice the last two operations are significantly slower than `rank/select` [17]. In particular, `open` is needed just to cover a border case where one node is an ancestor of the other in the Cartesian tree. Grossi and Ottaviano [18] replaced `open` by further `rank/selects` in this case, thus improving the time significantly.

Their formula [7, 18] represents the Cartesian tree using DFUDS [19]. In this paper we show that, if we use instead the BP representation for the tree [6], the RMQ formula can be considerably simplified because the border case does not need special treatment. The result is the fastest and most compact RMQ implementation so far: our structure uses $2.2n$ bits of space and answers RMQs in 1–4 microseconds. Current implementations in Simon Gog’s *SDSL* (<https://github.com/simongog/sdsl-lite>) and Giuseppe Ottaviano’s *Succinct* (<https://github.com/ot/succinct>) use $2.6n$ to $2.8n$ bits. Our implementation is also 3–4 times faster than that in *SDSL* and takes 50%–80% of the time of the implementation in *Succinct*. It is also 2–7 times faster than our own implementation of Fischer and Heun’s RMQ, while using less space.

State of the Art

Gabow et al. [1] showed that RMQs can be reduced to computing LCAs on a particular tree, called the *Cartesian tree* [2] of $A[1, n]$. This is a binary tree whose root is the position p of a minimum in $A[1, n]$ (the leftmost/rightmost one if we want that RMQs return the leftmost/rightmost minimum). Then its left and right children are the Cartesian trees of $A[1, p - 1]$ and $A[p + 1, n]$, respectively. Any cell $A[p]$ is thus represented by the Cartesian tree node with inorder position p , and it holds

$$\text{rmq}_A(i, j) = \text{inorder}(\text{lca}(\text{innode}(i), \text{innode}(j))), \quad (1)$$

where `inorder` and `innode` map from nodes to their inorder values and vice versa. Figure 1 shows an example array A and its Cartesian tree, and the translation of a query (ignore the other elements for now).

Later, Berkman and Vishkin [3] showed that the LCA problem on any tree can be reduced to an RMQ problem, on an array $D[1, 2n]$ containing the depths of the nodes traversed along an Eulerian tour on the tree: the LCA corresponds to the minimum in D between a cell of u and a cell of v in the array. Note that consecutive cells in D differ by ± 1 . Bender and Farach [4] represented those entries as a bitvector $E[1, 2n]$: $E[i] = 1$ if $D[i] - D[i - 1] = +1$ and $E[i] = 0$ if $D[i] - D[i - 1] = -1$, with $E[1] = 1$. On top of E , they gave a simple $O(1)$ -time solution to this restricted ± 1 -RMQ problem using $O(n)$ words of space. Figure 1 also shows this arrangement.

Therefore, one can convert an RMQ problem on A into an LCA problem on the Cartesian tree of A , then convert this problem into a ± 1 -RMQ problem on the depths of the Eulerian tour of the Cartesian tree, and finally solve this restricted ± 1 -RMQ problem in constant time. This solution requires $O(n)$ words of space.

Interestingly, the bitvector $E[1, 2n]$ used to answer LCA queries on a tree of n nodes defines the topology of the tree. If we traverse the tree in DFS order and write

Cartesian Tree

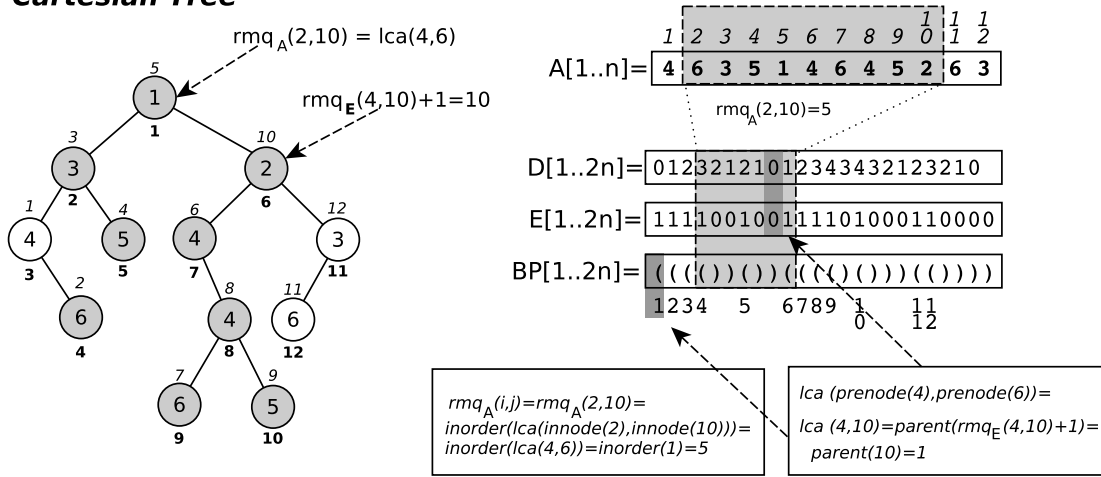


Figure 1: An example array $A[1, 12]$ (top right) and its Cartesian tree (left). We choose preorder numbers as node identifiers (in bold under the nodes), and also write inorder values on top of the nodes, in slanted font. The left rectangle on the bottom shows how query $\text{rmq}_A(2, 10)$ translates into query $\text{lca}(4, 6)$ on the Cartesian tree. We also show how this query, in turn, maps into $\text{rmq}_D(4, 10)$, on the array D of depths of the tree. Array E tells if consecutive entries of D increase or decrease, and is the same as a BP representation of the tree. The right rectangle on the bottom shows how query $\text{lca}(4, 10)$ is solved using $\text{rmq}_E(4, 10)$ and parent on the parentheses. This rmq_E query is a simpler ± 1 -RMQ problem. Now the nodes 4, 10, and 1 do not refer to preorders but to positions in BP, obtained from preorders with prenode . The corresponding preorder values are written below the BP array.

an opening parenthesis when we first arrive at a node and a closing one when we leave it, the resulting sequence of parentheses, $P[1, 2n]$, is exactly $E[1, 2n]$ if we interpret the opening parenthesis as a 1 and the closing one as a 0. In particular, consider the following two operations on bitvectors: $\text{rank}_b(E, i)$ is the number of bits equal to b in $E[1, i]$, and $\text{select}_b(E, j)$ is the position of the j th bit b in E . Both operations can be implemented in $O(1)$ time using just $o(n)$ additional bits on top of E [15, 16]. Then, if we identify a node x with the position of its opening parenthesis in P (which is a 1 in E), then the preorder position of x is $\text{preorder}(x) = \text{rank}_1(E, x)$, the node with preorder i is $\text{prenode}(i) = \text{select}_1(E, i)$, x is a leaf iff $E[x + 1] = 0$, and the depth of x is $D[x] = \text{rank}_1(E, x) - \text{rank}_0(E, x) = 2 \cdot \text{rank}_1(E, x) - x$.

This parentheses representation (called BP, for Balanced Parentheses) was indeed known, and it was even possible to navigate it in constant time by using just $2n + o(n)$ bits [6, 20]. This navigation was built on top of three primitives on parentheses: $\text{open}(x)/\text{close}(x)$ gave the position of the opening/closing parenthesis matching the closing/opening one at $P[x]$, and $\text{enclose}(x)$ gave the opening parenthesis position y so that $[y, \text{close}(y)]$ contained $P[x]$ most tightly. Many tree traversal operations are built on top of those primitives, for example the parent of x is $\text{parent}(x) = \text{enclose}(x)$, its next sibling is $\text{close}(x) + 1$ (if it exists), its first child is $x + 1$ (if it exists), its subtree size is $(\text{close}(x) - x + 1)/2$, x is an ancestor of y iff $x \leq y \leq \text{close}(x)$, etc.

Now, since E coincides with P , one could add the powerful lca operation to

the BP representation! Bender and Farach’s solution [4] applied on the bitvector $E[1, 2n]$ actually implements RMQs on the virtual array D . However, their ± 1 -RMQ solution used $O(n)$ words. Sadakane [5] improved their solution to use $O(n(\log \log n)^2 / \log n) = o(n)$ bits, and thus obtained a constant-time algorithm for $\text{lca}(x, y)$ on the BP representation (let $x < y$):

```

if  $y \leq \text{close}(x)$  then return  $x$ 
else return  $\text{parent}(\text{rmq}_E(x, y) + 1)$ 

```

where the first line addresses the special case where x is an ancestor of y , and rmq_E refers to the ± 1 -RMQ solution on $E[1, 2n]$. The rationale of the second line is that, since x and y descend from two distinct children of $z = \text{lca}(x, y)$, then $D[x, y]$ is minimized at the closing parenthesis that terminates each child of z , from the one that contains x to the one preceding that containing y . Adding 1 we get to the next sibling of that child, then we return its parent z . See Figure 1 once again.

Benoit et al. [19] presented an alternative format to represent a general tree using $2n$ parentheses, called DFUDS. We traverse the tree in DFS order, but this time, upon arriving for the first time to a node with d children, we write d opening parentheses and a closing one (in particular, a leaf is represented with a closing parenthesis). Nodes are identified with that closing parenthesis¹. It can be shown that the resulting sequence is also balanced if we append an artificial opening parenthesis at the beginning, and many traversal operations can be carried out with the primitives `open`, `close`, and `enclose`. In particular, we can directly arrive at the i th child of x with $\text{next}_0((\text{close}(x - i) + 1))$, where $\text{next}_0(t) = \text{select}_0(\text{rank}_0(t - 1) + 1)$ finds the first 0 from t . The number of children of x can be computed as $d = x - \text{prev}_0(x) + 1$, where $\text{prev}_0(t) = \text{select}_0(\text{rank}_0(t - 1))$ finds the last 0 before t . In DFUDS, nodes are also listed in preorder, and there is a closing parenthesis terminating each, thus $\text{preorder}(x) = \text{rank}_0(E, x)$.

Jansson et al. [8] showed that $\text{lca}(x, y)$ can also be computed on the DFUDS representation, as follows (let $x < y$):

```

return  $\text{parent}(\text{next}_0(\text{rmq}_E(x, y - 1) + 1))$ ,

```

where no check for ancestorship is needed². The rationale is similar as before: since in DFUDS D decreases by 1 along each subtree area, $\text{rmq}_E(x, y - 1)$ finds the final closing parenthesis of the child of $z = \text{lca}(x, y)$ that precedes the one containing y . Adding 1 and finding the parent gives z . The formula for $\text{parent}(w)$ in DFUDS is $\text{next}_0(\text{open}(\text{prev}_0(w)))$. Figure 2 shows our example, now on DFUDS. The formula with DFUDS turns out to be simpler than with BP.

Now we could represent a tree of n nodes in $2n + o(n)$ bits and compute lca on it in constant time, and Eq. (1) allowed us to convert rmq_A into an lca operation on its Cartesian tree. It seems that the road to constant-time rmq_A using just the $2n + o(n)$ bits of its Cartesian tree, and without accessing A , was paved! However, there was still a problem: how to support the operations `inorder` and `innode` on the Cartesian

¹In some cases, the first opening parenthesis is used, but the closing one is more convenient here.

²The check is present in their paper, but it is unnecessary (K. Sadakane, personal communication).

Cartesian Tree

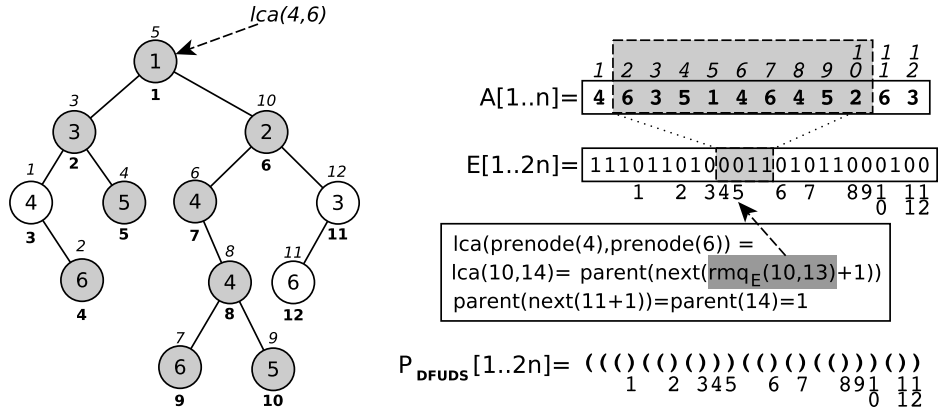


Figure 2: The same arrangement of Figure 1, now on the DFUDS representation of the Cartesian tree. The query $\text{rmq}_A(2, 10)$ becomes $\text{lca}(4, 6)$, which we translate into $\text{lca}(10, 14)$ when the node identifiers become positions in DFUDS instead of preorders (the translation is shown on the bottom of the sequence P_{DFUDS}).

tree. Sadakane [5] had solved the problem on suffix trees, but in his case the tree had exactly one leaf per entry in A , so he only needed to find the i th leaf, and this could be done by extending $\text{rank}/\text{select}$ operations to find 10s (BP) or 00s (DFUDS) in E . In the general case, one could add artificial leaves to every node, but this would increase the space to $4n + o(n)$ bits.

Fischer and Heun [7] found a solution that used just $2n + o(n)$ bits, which also turned out to be asymptotically optimal. The idea is to use a known isomorphism (see, e.g., [6]) between binary trees of n nodes and general ordinal trees of $n + 1$ nodes: We create an extra root for the general tree, and its children are the nodes in the leftmost path of the binary tree. Recursively, the right subtree of each node x in the leftmost path is converted into a general tree, using x as its extra root. A key property of this transformation is that inorders in the binary tree become preorders (plus 1) in the general tree. As seen, we can easily map between nodes and their preorders in general trees. Figure 3 continues our example.

However, the lca in the Cartesian tree (which is what we want) is not the same lca in the resulting general tree; some adjustments are necessary. Fischer and Heun chose to use DFUDS for their $\text{rmq}_A(i, j)$ solution, where it turns out that the adjustments to use a general tree actually remove the need to compute parent , but add back the need to check for ancestorship:

$$\begin{aligned}
 w &\leftarrow \text{rmq}_E(\text{select}_0(i + 1), \text{select}_0(j)) \\
 \text{if } \text{rank}_0(\text{open}(w)) &= i \text{ then return } i \\
 \text{else return } \text{rank}_0(w) &
 \end{aligned}
 \tag{2}$$

The select_0 operations find the nodes with preorder i and $j - 1$ (recall there is an extra root with preorder 1), then w is the position of the closing parenthesis of the result. The next line verifies that x is not an ancestor of y , and the last line returns

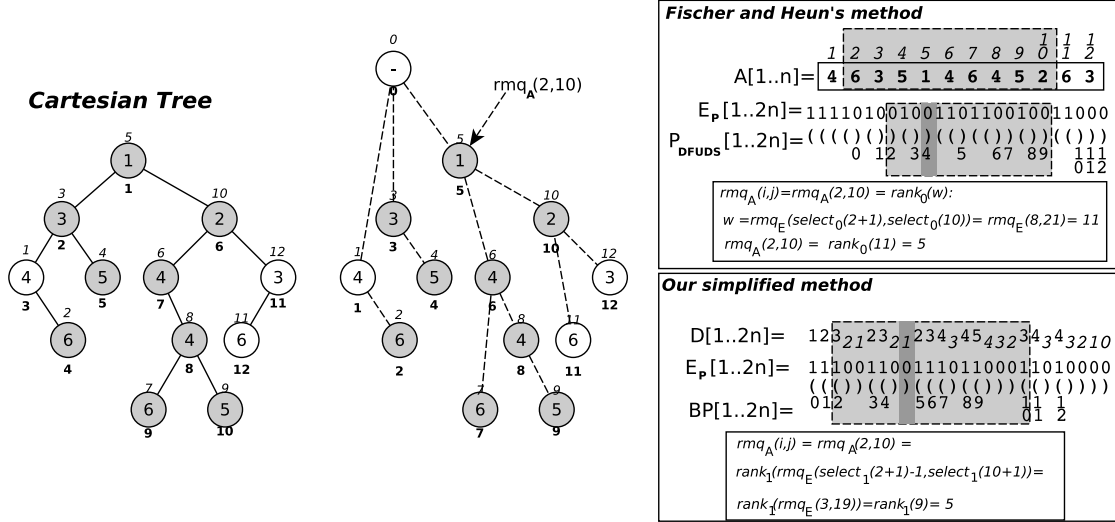


Figure 3: The general tree derived from the example Cartesian tree. Note how inorder numbers of the binary Cartesian tree became preorder numbers in the general tree (we start preorders from 0 to help see the mapping). On the right, the formulas used by Fischer and Heun based on DFUDS (on the top) and the one proposed in this paper, based on BP (on the bottom). To reuse the same isomorphism of Fischer and Heun, we illustrate the variant of our formula that uses the leftmost path of the tree as the root children.

the corresponding preorder value. For this formula to be correct, it is necessary that rmq_E returns the position of the leftmost minimum. Figure 3 (top left) shows a query.

Grossi and Ottaviano [18] replaced the ancestorship test by one that does not use the costly `open` operation:

$$\begin{aligned}
 w &\leftarrow rmq_E(select_0(i+1), select_0(j)) \\
 \text{if } D[select_0(i)+1] &\leq D[w-1] \text{ then return } i \\
 \text{else return } rank_0(w) &
 \end{aligned} \tag{3}$$

where as explained we can compute $D[k] = 2 \cdot rank_1(E, k) - k$.

A Simplified Implementation

The current implementations of rmq_A build on the DFUDS representation of the general tree derived from the Cartesian tree, and follow either the formula of Fischer and Heun [7] (Eq. (2), in *SDSL*), or that of Grossi and Ottaviano [18] (Eq. (3), in *Succinct*). We show that, if we use the BP representation instead of DFUDS, we obtain a simpler formula. Let us assume, as before, that rmq_E returns the leftmost minimum. Then, our conversion from the binary Cartesian tree into a general tree must go in the opposite direction: the children of the extra root are the nodes in the *rightmost* path of the binary tree, and so on recursively. With this representation, it turns out that a correct formula is

$$rmq_A(i, j) = rank_0(rmq_E(select_0(i), select_0(j))) \tag{4}$$

where no checks for ancestorship are necessary. Now we prove this formula is correct.

Lemma 1 *On a rightmost-path general tree built from the Cartesian tree of A , Eq. (4) holds.*

Proof. On the rightmost-path representation, the binary tree node with inorder i becomes the general tree node with *postorder* i , which easily seen by induction. The closing parentheses of nodes x and y , which have postorders i and j , are thus found with $p = \text{select}_0(i)$ and $q = \text{select}_0(j)$. Now let $z = \text{lca}(x, y)$. Then, in the Cartesian tree, x descends from the left child of z , z_l , and y descends from the right child, z_r . In the general tree, z_l is the first child of z , whereas z_r is its next sibling. Therefore the closing parenthesis of z , at position r , is between p and q . Further, y descends from some sibling z' to the right of z . Between p and q , the minima in D occur at the closing parentheses of z and of its siblings to the right, up to (but not including) z' . Thus the leftmost of those positions is precisely r , where z closes. Finally, $\text{rank}_0(r)$ is the postorder position of z , and the inorder position of the cell in A .

The formula also works if y descends from x in the Cartesian tree. Since $i < j$, the inorder of x is smaller than the inorder of y , and thus y can only descend from the right child of x . Then the first minima in $[p, q]$ is precisely p , the closing parenthesis of x , and thus $z = x$. \square

If we want to use the leftmost-path mapping, we need that rmq_E returns the rightmost minimum position in the range. In this case, it holds

$$\text{rmq}_A(i, j) = \text{rank}_1(\text{rmq}_E(\text{select}_1(i + 1) - 1, \text{select}_1(j + 1))).$$

In this case, we must subtract 1 from p (which is now the position where node x opens) to ensure that the rightmost minimum in $[p - 1, q]$ is actually $p - 1$ when y descends from x . Figure 3 (bottom right) shows a query.

In the next section we show that our formula yields a significant time reduction compared to DFUDS-based ones.

Experimental Results

Our implementation of the formula in Eq. (4) uses RMM-trees [9, 17], where only the field *min* is needed on the RMM-tree nodes in order to compute rmq_E . The operations **rank** and **select** are implemented by ourselves, using one level of counters for **rank** and binary search for **select**.

We compare our implementation with those in *SDSL* and *Succinct*, which are based on DFUDS (Eqs. (2) and (3), respectively). As a control, we also implement ourselves the DFUDS-based solution of Eq. (2) using RMM-trees and our **rank/select** components; this is called *DFUDS* in our charts.

Our first experiment compares the four implementations on randomly generated arrays A of sizes from $n = 10^4$ to $n = 10^{10}$, with randomly chosen ranges $[i, j]$ of fixed length 10,000. Figure 4 shows the results (*Succinct* did not build on the largest arrays). Our implementation uses almost always 2.2 bits per element (bpe), that is, 0.2 on top of the 2 bpe needed by the BP (or DFUDS) representation. Our DFUDS implementation, instead, increases the space because the average excess grows with n

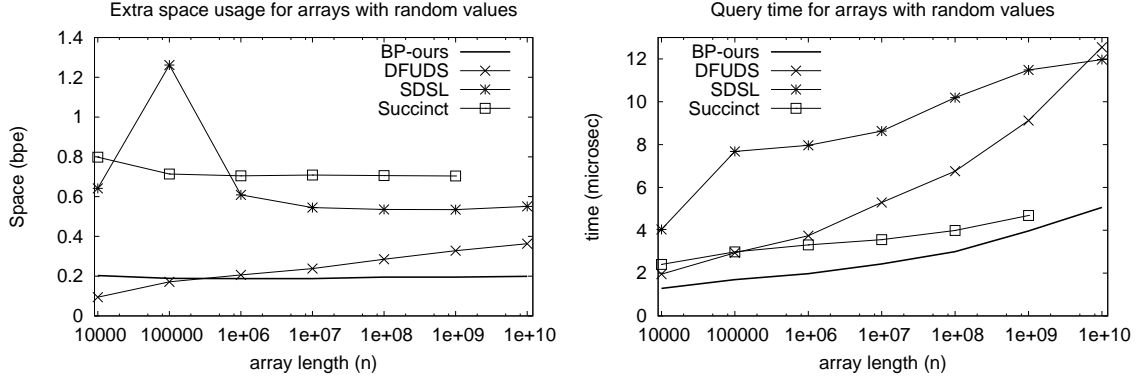


Figure 4: Query space and time on random arrays, for ranges of size 10,000.

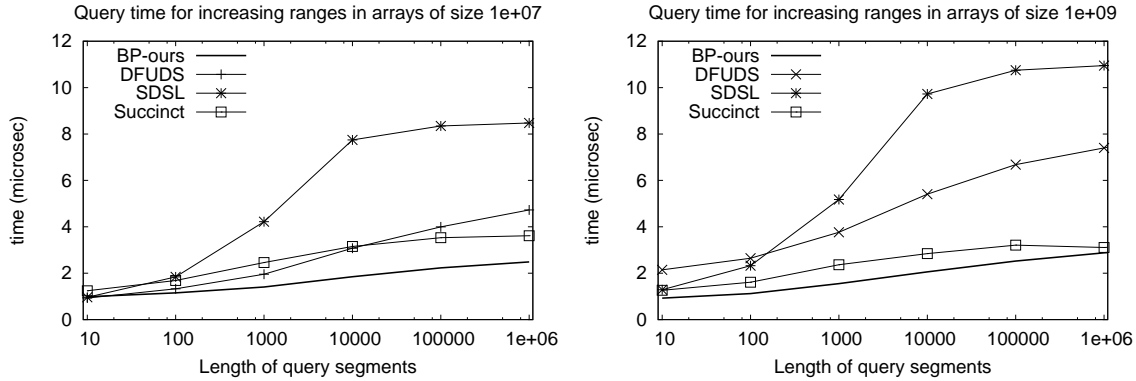


Figure 5: Query time on random arrays, for ranges of increasing size and two values of n .

in this format, and thus the RMM-tree counters need more bits. The implementations in *SDSL* and *Succinct* use at least 2.6–2.8 bpe. Our solution is also the fastest, taking 1–4 microseconds (μsec) per query as n grows. It is followed by *Succinct* and, far away, by *SDSL*. Our *DFUDS* implementation is fast for short arrays, but it becomes slower when n grows. This is probably because operation `open` matches a farther parenthesis as n grows.

Figure 5 shows how the times are affected by the size of the query range. As it can be seen, our implementation and *Succinct* show a very slow increase, whereas times grow much faster in *SDSL* and *DFUDS*. This may be due to the `open` operation, whose time grows in practice with the distance to its parent. Larger intervals return nodes closer to the root, whose former siblings are larger, and so is the distance to the parent in *DFUDS*.

Our final experiment measures the effect of the order in A on the space and time of the structures. Given a parameter Δ , our entry $A[i]$ is chosen at random in $[i - \Delta, i + \Delta]$, or in $[n - i - \Delta, n - i + \Delta]$, thus the smaller Δ , the more sorted is A in increasing/decreasing order. Figure 6 shows the results.

Our implementation maps the leftmost path of the Cartesian tree to the children

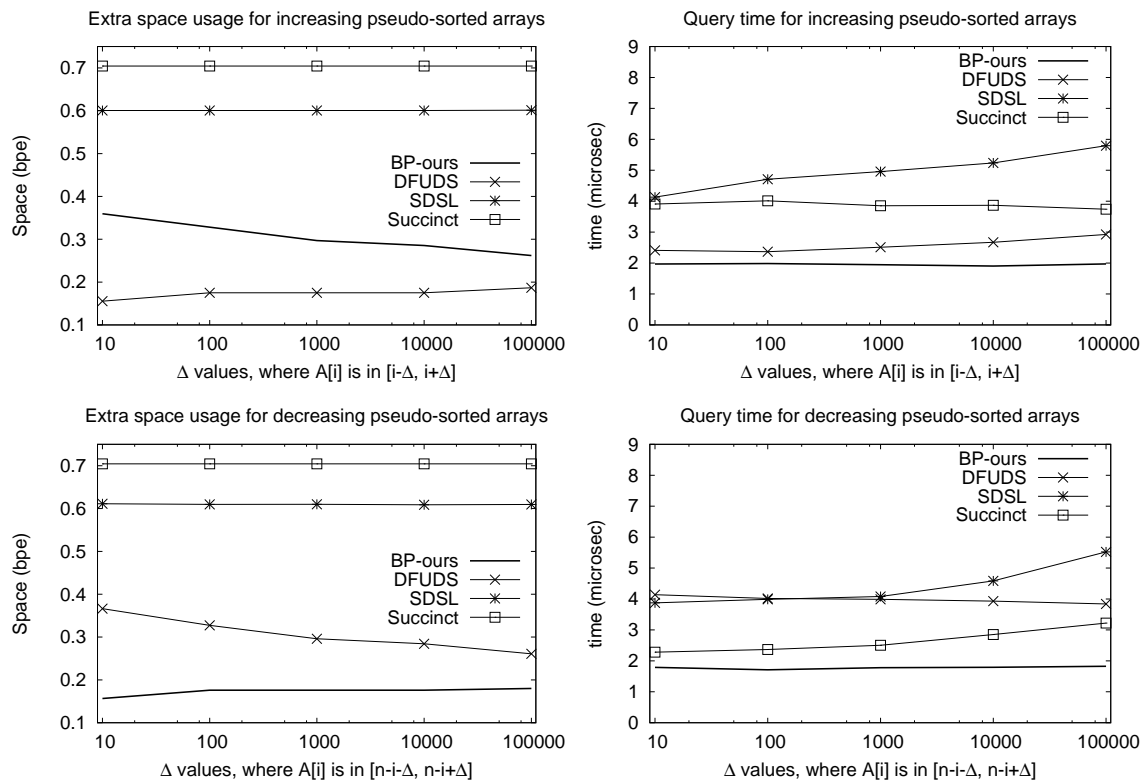


Figure 6: Query time on pseudo-sorted arrays, $n = 10^6$ and ranges of size 10,000.

of the general tree. As a result, the structure takes more space and time when the array is more sharply increasing, because the general tree is deeper and the RMM-tree stores larger values. Instead, it does not change much when A is decreasing (one could use one mapping or the other as desired, since we know A at construction time). *DFUDS* shows the opposite effect, because the *DFUDS* excesses are smaller when the tree is deeper. It is not clear how can one use the rightmost-path mapping in the case of *DFUDS*, however, as it is not symmetric. The space of *SDSL* and *Succinct* is not affected at all by the lack of randomness, but *SDSL* turns out to be faster on less random arrays, regardless of whether they are increasing or decreasing.

Conclusions

We have presented an alternative design to Fischer and Heun's RMQ solution that uses $2n + o(n)$ bits and constant time [7]. Our implementation uses $2.2n$ bits and takes 1–4 microseconds per query. This is noticeably smaller and faster than the current implementations in libraries *SDSL* and *Succinct*, which follow Fischer and Heun's design. By using BP instead of *DFUDS* succinct tree representation, our RMQ formula simplifies considerably. We have left our implementation publicly available at <https://github.com/hferrada/rmq.git>, and our *DFUDS*-based implementation at <https://github.com/hferrada/rmqFischerDFUDS.git>.

Any ± 1 -RMQ implementation can be used together with our new formula. Our

current implementation of ± 1 -RMQs is not formally constant time, as it builds on RMM-trees [9, 17]. Although truly constant-time solutions are not promising in practice [5, 9], and we have shown that the time of RMM-trees grows very slowly with n , it would be interesting to devise a practical and constant-time solution.

References

- [1] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, “Scaling and related techniques for geometry problems,” in *Proc. 16th STOC*, 1984, pp. 135–143.
- [2] J. Vuillemin, “A unifying look at data structures,” *Comm. ACM*, vol. 23, no. 4, pp. 229–239, 1980.
- [3] O. Berkman and U. Vishkin, “Recursive star-tree parallel data structure,” *SIAM J. Comp.*, vol. 22, no. 2, pp. 221–242, 1993.
- [4] M. Bender and M. Farach-Colton, “The LCA problem revisited,” in *Proc. 4th LATIN*, ser. LNCS 1776, 2000, pp. 88–94.
- [5] K. Sadakane, “Compressed suffix trees with full functionality,” *Theor. Comp. Syst.*, vol. 41, no. 4, pp. 589–607, 2007.
- [6] J. I. Munro and V. Raman, “Succinct representation of balanced parentheses and static trees,” *SIAM J. Comp.*, vol. 31, no. 3, pp. 762–776, 2001.
- [7] J. Fischer and V. Heun, “Space-efficient preprocessing schemes for range minimum queries on static arrays,” *SIAM J. Comp.*, vol. 40, no. 2, pp. 465–492, 2011.
- [8] J. Jansson, K. Sadakane, and W.-K. Sung, “Ultra-succinct representation of ordered trees with applications,” *J. Comp. Sys. Sci.*, vol. 78, no. 2, pp. 619–631, 2012.
- [9] G. Navarro and K. Sadakane, “Fully-functional static and dynamic succinct trees,” *ACM Trans. Alg.*, vol. 10, no. 3, p. article 16, 2014.
- [10] J. Fischer, V. Mäkinen, and G. Navarro, “Faster entropy-bounded compressed suffix trees,” *Theor. Comp. Sci.*, vol. 410, no. 51, pp. 5354–5364, 2009.
- [11] K. Sadakane, “Succinct data structures for flexible text retrieval systems,” *J. Discr. Alg.*, vol. 5, pp. 12–22, 2007.
- [12] R. Konow and G. Navarro, “Faster compact top-k document retrieval,” in *Proc. 23rd DCC*, 2013, pp. 351–360.
- [13] G. Navarro, Y. Nekrich, and L. M. S. Russo, “Space-efficient data-analysis queries on grids,” *Theor. Comp. Sci.*, vol. 482, pp. 60–72, 2013.
- [14] G. Chen, S. J. Puglisi, and W. F. Smyth, “Lempel-Ziv factorization using less time & space,” *Mathematics in Computer Science*, vol. 1, pp. 605–623, 2008.
- [15] G. Jacobson, “Space-efficient static trees and graphs,” in *Proc. 30th FOCS*, 1989, pp. 549–554.
- [16] D. Clark, “Compact PAT trees,” Ph.D. dissertation, Univ. of Waterloo, Canada, 1996.
- [17] D. Arroyuelo, R. Cánovas, G. Navarro, and K. Sadakane, “Succinct trees in practice,” in *Proc. 12th ALENEX*, 2010, pp. 84–97.
- [18] R. Grossi and G. Ottaviano, “Design of practical succinct data structures for large data collections,” in *Proc. 12th SEA*, ser. LNCS 7933, 2013, pp. 5–17.
- [19] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao, “Representing trees of higher degree,” *Algorithmica*, vol. 43, no. 4, pp. 275–292, 2005.
- [20] R. F. Geary, N. Rahman, R. Raman, and V. Raman, “A simple optimal representation for balanced parentheses,” *Theor. Comp. Sci.*, vol. 368, no. 3, pp. 231–246, 2006.