Journal of Discrete Algorithms ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Improved Range Minimum Queries ☆

Héctor Ferrada, Gonzalo Navarro *

Center of Biotechnology and Bioengineering, Department of Computer Science, University of Chile, Santiago, Chile

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Range Minimum Queries
Lowest common ancestors
Cartesian trees
Balanced Parentheses
Compact data structures

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. We also improve the current implementation of the BP representation for this purpose. This leads to the fastest and most compact practical implementation to date.

1. 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 $1 \le i \le j \le n$ and returns

$$\mathsf{rmq}_A(i,j) = \mathrm{argmin}_{i \le k \le j} A[k],$$

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 $\mathsf{lca}(u,v)$, the lowest node that is an ancestor of both u and v. Gabow et al. [10] showed that RMQs can be reduced to computing LCAs on a particular tree, called the *Cartesian tree* [22] of $A[1,n]$. Later, Berkman and Vishkin [4] 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 [2] then gave a solution for this so-called ± 1-RMQ problem in constant time and linear space (i.e., $O(n)$ words). Sadakane [20] 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 [17]). Finally, Fischer and Heun [8] 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 leads 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 [20,15,19], suffix trees [20,9], document retrieval [21,16], two-dimensional grids [18], Lempel–Ziv parsing [5], etc.

Their RMQ computation [8] uses three kinds of operations: several rank/selects on bitvectors [14,6], one ± 1-RMQ [2], and one open on parentheses [17]. Although all can be implemented in constant time, in practice the last two operations are significantly slower than rank/select [1]. In particular, open is needed just to cover a border case where one node is an

☆ Funded with basal funds FB0001, CONICYT, Chile. An early version of this article appeared in *Proc. Data Compression Conference 2016*.
* Corresponding author.
E-mail addresses: hferrada@dcc.uchile.cl (H. Ferrada), gnavarro@dcc.uchile.cl (G. Navarro).

http://dx.doi.org/10.1016/j.jda.2016.09.002
1570-8667/

Please cite this article in press as: H. Ferrada, G. Navarro, Improved Range Minimum Queries, J. Discret. Algorithms (2016), http://dx.doi.org/10.1016/j.jda.2016.09.002

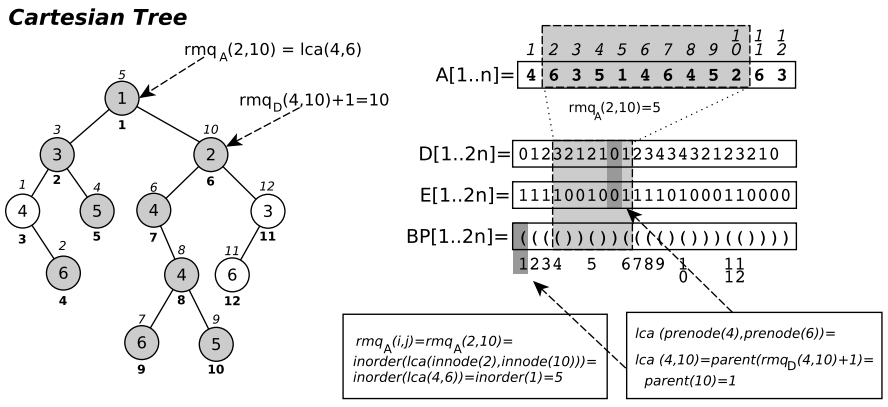

Fig. 1. An example array $A[1, 12]$ (top right) and its Cartesian tree (left). We choose preorder numbers as node identifiers (in bold under the nodes), and also write inorder values on top of the nodes, in slanted font. The left rectangle on the bottom shows how query $\mathsf{rmq}_A(2, 10)$ translates into query $\mathsf{lca}(4, 6)$ on the Cartesian tree. We also show how this query, in turn, maps into $\mathsf{rmq}_D(4, 10)$, 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 $\mathsf{lca}(4, 10)$ is solved using $\mathsf{rmq}_D(4, 10)$ and parent on the parentheses. This rmq_D 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.

ancestor of the other in the Cartesian tree. Grossi and Ottaviano [13] replaced open by further rank/selects in this case, thus improving the time significantly.

Their formula [8,13] represents the Cartesian tree using DFUDS [3]. In this paper we show that, if we use instead the BP representation for the tree [17], the RMQ formula can be considerably simplified because the border case does not need special treatment. In addition, we improve the current implementations of the BP representation, tailoring them to solve RMQs. The result is the fastest and most compact RMQ implementation so far: our structure uses $2.1n$ bits of space and answers RMQs in 1–3 microseconds. Current implementations in Simon Gog's *SDSL* [12] (https://github.com/simongog/sdsl-lite) and Giuseppe Ottaviano's *Succinct* [13] (https://github.com/ot/succinct) use from $2.6n$ to $2.8n$ bits. Our implementation is also 3–6 times faster than that in *SDSL* and twice as fast as the implementation in *Succinct*. It is also 2–4 times faster than our own implementation of Fischer and Heun's RMQ, while using less space.

2. State of the art

Gabow et al. [10] showed that RMQs can be reduced to computing LCAs on a particular tree, called the *Cartesian tree* [22] 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

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

where inorder and innode map from nodes to their inorder values and vice versa. Fig. 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 [4] 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 [2] 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. Fig. 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 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: $\mathsf{rank}_b(E, i)$ is the number of bits equal to b in $E[1, i]$, and $\mathsf{select}_b(E, j)$ is the position of the jth bit b in E. Both operations can be implemented in $O(1)$ time using just $o(n)$ additional bits on top of E [14,6]. 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 $\mathsf{preorder}(x) = \mathsf{rank}_1(E, x)$, the node with preorder i is $\mathsf{prenode}(i) = \mathsf{select}_1(E, i)$, x is a leaf iff $E[x+1] = 0$, and the depth of x is $D[x] = \mathsf{rank}_1(E, x) - \mathsf{rank}_0(E, x) = 2 \cdot \mathsf{rank}_1(E, x) - x$.

Please cite this article in press as: H. Ferrada, G. Navarro, Improved Range Minimum Queries, J. Discret. Algorithms (2016), http://dx.doi.org/10.1016/j.jda.2016.09.002