

9

Range-Minimum Queries

14 December 2023

Sebastian Wild

Learning Outcomes

1. Know the *RMQ problem* and its *connection* to longest common extensions in strings.
2. Know and understand trivial RMQ solutions and *sparse tables*.
3. Know and understand the *Cartesian trees* data structure.
4. Know and understand the *exhaustive-tabulation technique* for RMQ with linear-time preprocessing.

Unit 9: Range-Minimum Queries



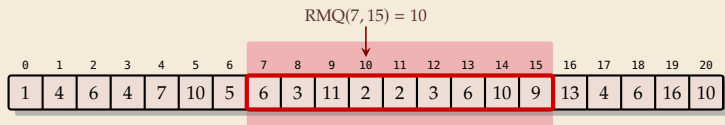
9 Range-Minimum Queries

- 9.1 Introduction
- 9.2 RMQ, LCP, LCE, LCA — WTF?
- 9.3 Trivial Solutions & Sparse Tables
- 9.4 Cartesian Trees
- 9.5 Exhaustive Tabulation

9.1 Introduction

Range-minimum queries (RMQ)

- array/numbers don't change
- ▶ **Given:** Static array $A[0..n)$ of numbers
 - ▶ **Goal:** Find minimum in a range;
 A known in advance and can be preprocessed



- ▶ **Nitpicks:**
 - ▶ Report *index* of minimum, not its value
 - ▶ Report *leftmost* position in case of ties

Rules of the Game

- ▶ comparison-based \rightsquigarrow values don't matter, only relative order
- ▶ Two main quantities of interest:
 1. **Preprocessing time:** Running time $P(n)$ of the preprocessing step \rightsquigarrow space usage $\leq P(n)$
 2. **Query time:** Running time $Q(n)$ of one query (using precomputed data)
- ▶ Write $\langle P(n), Q(n) \rangle$ **time solution** for short

9.2 RMQ, LCP, LCE, LCA — WTF?

Recall Unit 6

Application 4: Longest Common Extensions

- ▶ We implicitly used a special case of a more general, versatile idea:

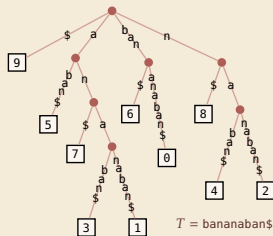
The *longest common extension (LCE)* data structure:

- ▶ **Given:** String $T[0..n-1]$
- ▶ **Goal:** Answer LCE queries, i.e.,
 given positions i, j in T ,
 how far can we read the same text from there?
 formally: $LCE(i, j) = \max\{\ell : T[i..i + \ell] = T[j..j + \ell]\}$

↪ use suffix tree of T !

- ▶ In \mathcal{T} : $LCE(i, j) = \overset{\text{longest common prefix of } i\text{th and } j\text{th suffix}}{\text{LCP}(T_i, T_j)} \rightsquigarrow \text{same thing, different name!}$
 $= \text{string depth of}$
 $\text{lowest common ancestor (LCA) of}$
 leaves \boxed{i} and \boxed{j}



- ▶ in short: $LCE(i, j) = LCP(T_i, T_j) = \text{stringDepth}(\text{LCA}(\boxed{i}, \boxed{j}))$



Recall Unit 6

Efficient LCA

How to find lowest common ancestors?

- ▶ Could walk up the tree to find LCA $\rightsquigarrow \Theta(n)$ worst case 
- ▶ Could store all LCAs in big table $\rightsquigarrow \Theta(n^2)$ space and preprocessing 



Amazing result: Can compute data structure in $\Theta(n)$ time and space that finds any LCA is **constant(!) time**.

- ▶ a bit tricky to understand
- ▶ but a theoretical breakthrough
- ▶ and useful in practice

and suffix tree construction inside ...



\rightsquigarrow for now, use $O(1)$ LCA as black box.

\rightsquigarrow After linear preprocessing (time & space), we can find LCEs in $O(1)$ time.

Finally: Longest common extensions

- ▶ In Unit 6: Left question open how to compute LCA in suffix trees
- ▶ But: Enhanced Suffix Array makes life easier!

$$\text{LCE}(i, j) = \text{LCP}[\text{RMQ}_{\text{LCP}}(\min\{R[i], R[j]\} + 1, \max\{R[i], R[j]\})]$$

Inverse suffix array: going left & right

▶ to understand the fastest algorithm, it is helpful to define the *inverse suffix array*.

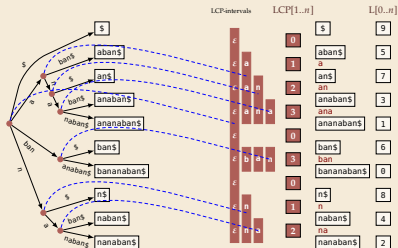
- ▶ $R[i] = r \iff L[r] = i$ $L = \text{leaf array}$
- \iff there are r suffixes that come before T_i in sorted order
- $\iff T_i$ has (0-based) *rank* $r \rightsquigarrow$ call $R[0..n]$ the *rank array*

i	$R[i]$	T_i		r	$L[r]$	$T_{i[r]}$
0	6 th	bananabans\$	right $R[0] = 6$	0	9	\$
1	4 th	ananabans\$		1	5	abans\$
2	9 th	nanabans\$		2	7	ans\$
3	3 th	anabans\$		3	3	anabans\$
4	8 th	nabans\$		4	1	ananabans\$
5	1 th	abans\$		5	6	ban\$
6	5 th	bans\$		6	0	bananabans\$
7	2 th	ans\$		7	8	n\$
8	7 th	n\$	left $L[8] = 4$	8	4	nabans\$
9	0 th	\$		9	2	nanabans\$

sort suffixes

25

LCP array and internal nodes



\rightsquigarrow Leaf array $L[0..n]$ plus LCP array $LCP[1..n]$ encode full tree!

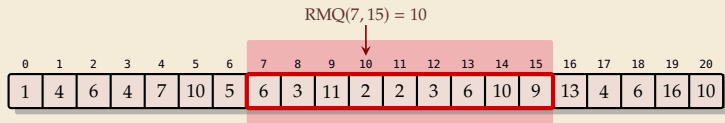
35

RMQ Implications for LCE

- ▶ Recall: Can compute (inverse) suffix array and LCP array in $O(n)$ time
- ↪ A $\langle P(n), Q(n) \rangle$ time RMQ data structure implies a $\langle P(n), Q(n) \rangle$ time solution for longest-common extensions

9.3 Trivial Solutions & Sparse Tables

Trivial Solutions



- ▶ Two easy solutions show extreme ends of scale:

1. Scan on demand

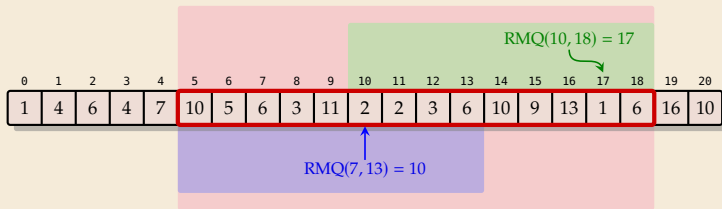
- ▶ no preprocessing at all
 - ▶ answer $\text{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min
- $\rightsquigarrow \langle O(1), O(n) \rangle$

2. Precompute all

- ▶ Precompute all answers in a big 2D array $M[0..n][0..n)$
 - ▶ queries simple: $\text{RMQ}(i, j) = M[i][j]$
- $\rightsquigarrow \langle O(n^3), O(1) \rangle$
- ▶ Preprocessing can reuse partial results $\rightsquigarrow \langle O(n^2), O(1) \rangle$

Sparse Table

- ▶ **Idea:** Like “precompute-all”, but keep only some entries
- ▶ store $M[i][j]$ iff $\ell = j - i + 1$ is 2^k .
 - ↪ $\leq n \cdot \lg n$ entries
 - ↪ Can be stored as $M'[i][k]$
- ▶ How to answer queries?



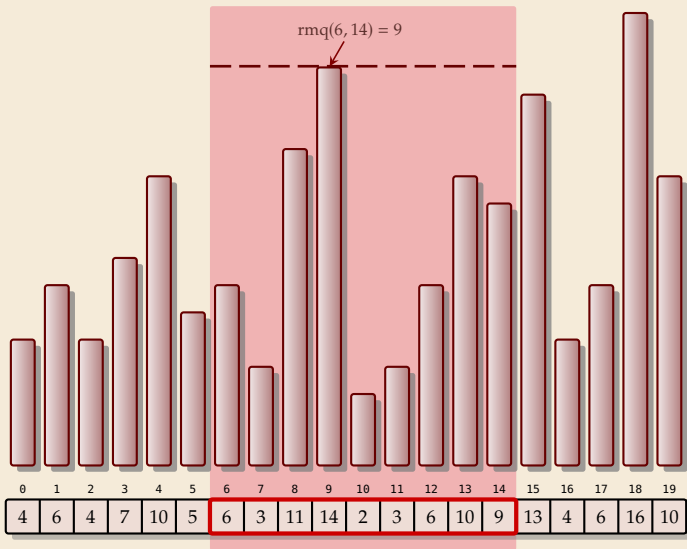
- ▶ Preprocessing can be done in $O(n \log n)$ times

↪ $\langle O(n \log n), O(1) \rangle$ time solution!

1. Find k with $\ell/2 \leq 2^k \leq \ell$
2. Cover range $[i..j]$ by 2^k positions right from i and 2^k positions left from j
3. $RMQ(i, j) = \arg \min\{A[rmq_1], A[rmq_2]\}$
with $rmq_1 = RMQ(i, i + 2^k - 1)$
 $rmq_2 = RMQ(j - 2^k + 1, j)$

9.4 Cartesian Trees

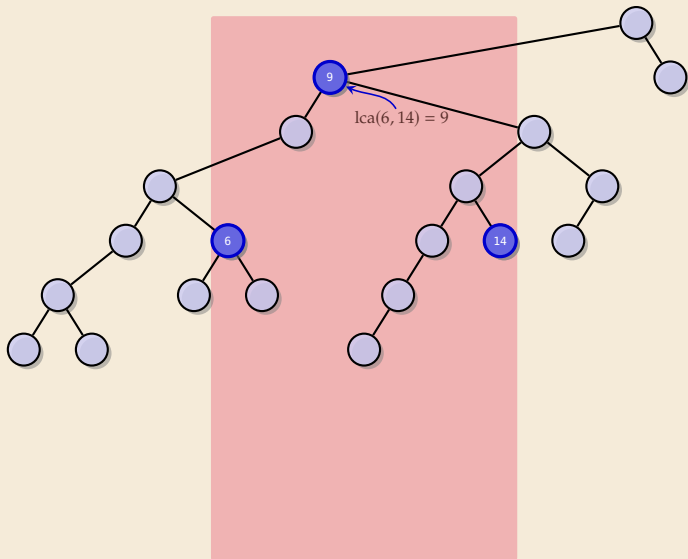
RMQ & LCA



- ▶ **Range-max queries** on array A :
$$\text{rmq}_A(i, j) = \arg \max_{i \leq k \leq j} A[k]$$

$$= \text{index of max}$$
- ▶ **Task:** Preprocess A ,
then answer RMQs fast
ideally constant time!

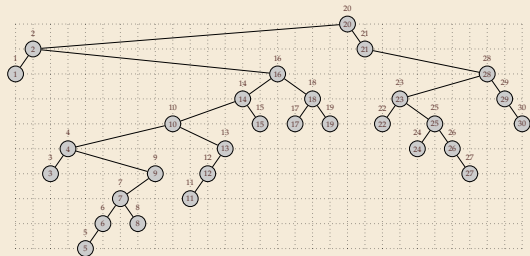
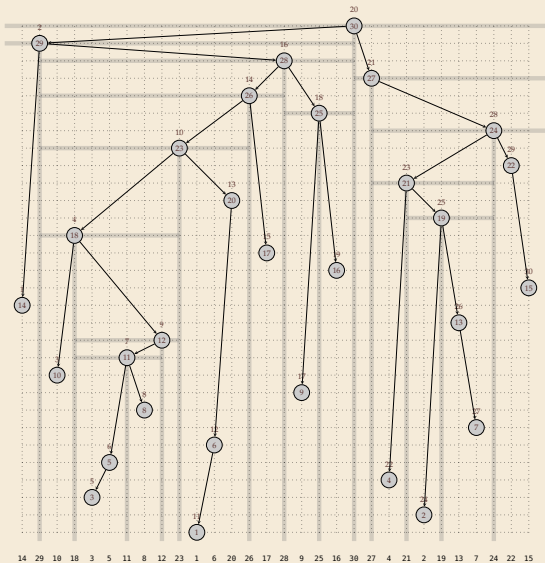
RMQ & LCA



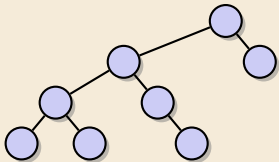
- ▶ **Range-max queries** on array A :
$$\text{rmq}_A(i, j) = \arg \max_{i \leq k \leq j} A[k]$$

$$= \text{index of max}$$
- ▶ **Task:** Preprocess A ,
then answer RMQs fast
ideally constant time!
- ▶ **Cartesian tree:** (cf. *treap*)
construct binary tree by
sweeping line down
- ▶ $\text{rmq}(i, j) = \text{inorder of}$
lowest common ancestor (LCA)
of i th and j th node in inorder

Cartesian Tree – Larger Example



Counting binary trees



- ▶ Given the Cartesian tree, all RMQ answers are determined and vice versa!

- ▶ How many different Cartesian trees are there for arrays of length n ?

- ▶ known result: *Catalan numbers* $\frac{1}{n+1} \binom{2n}{n}$

- ▶ easy to see: $\leq 2^{2n}$

↪ many arrays will give rise to the same Cartesian tree

Can we exploit that?

9.5 Exhaustive Tabulation

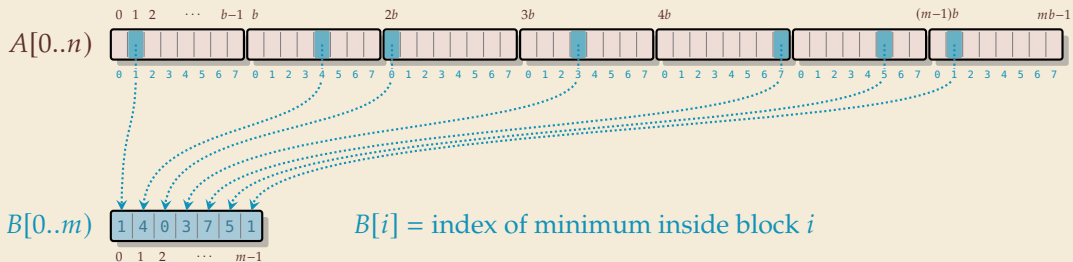
Four Russians?

The exhaustive-tabulation technique to follow is often called “Four Russians trick” . . .

- ▶ The algorithmic technique was published 1970 by V. L. Arlazarov, E. A. Dinitz, M. A. Kronrod, and I. A. Faradžev
- ▶ all worked in Moscow at that time . . . but not even clear if all are Russians!
(Arlazarov and Kronrod are Russian)
- ▶ American authors coined the slightly derogatory “Method of Four Russians” . . . name in widespread use

Bootstrapping

- ▶ We know a $\langle O(n \log n), O(1) \rangle$ time solution
- ▶ If we use that for $m = \Theta(n/\log n)$ elements, $O(m \log m) = O(n)$!
- ▶ Break A into blocks of $b = \lceil \frac{1}{4} \lg n \rceil$ numbers
- ▶ Create array of block minima $B[0..m]$ for $m = \lceil n/b \rceil = O(n/\log n)$

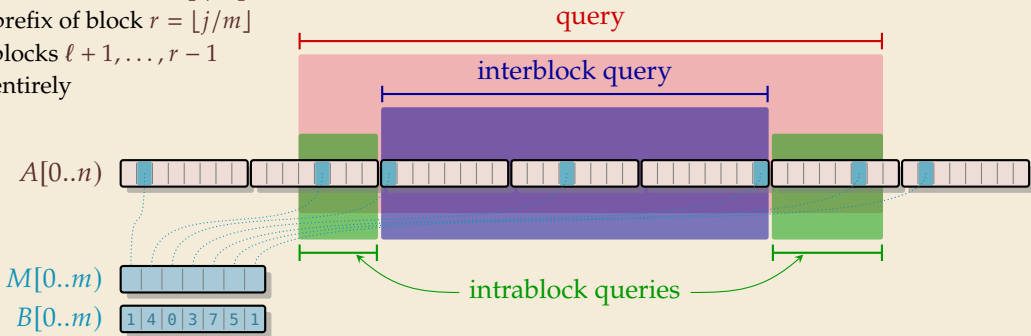


↪ Use sparse tables for B .

↪ Can solve RMQs in $B[0..m]$ in $\langle O(n), O(1) \rangle$ time

Query decomposition

- ▶ Query $\text{RMQ}_A(i, j)$ covers
 - ▶ suffix of block $\ell = \lfloor i/m \rfloor$
 - ▶ prefix of block $r = \lfloor j/m \rfloor$
 - ▶ blocks $\ell + 1, \dots, r - 1$ entirely



▶ $\text{RMQ}_A(i, j) = \arg \min_{k \in K} A[k]$ with $K = \left\{ \begin{array}{l} \text{RMQ}_{\text{block } \ell}(i - \ell b, (\ell + 1)b - 1), \\ b \cdot \text{RMQ}_M(\ell + 1, r - 1) + \\ B[\text{RMQ}_M(\ell + 1, r - 1)], \\ \text{RMQ}_{\text{block } r}(rb, j - rb) \end{array} \right\}$

↪ only 3 possible values to check
if **intrablock** and **interblock** queries known ✓

Intrablock queries [1]

↪ It remains to solve the **intrablock** queries!

▶ Want $\langle O(n), O(1) \rangle$ time overall

↖ must include preprocessing for all $m = \left\lceil \frac{n}{b} \right\rceil = \Theta\left(\frac{n}{\log n}\right)$ blocks!

▶ many blocks, but just $b = \left\lceil \frac{1}{4} \lg n \right\rceil$ numbers long

↪ Cartesian tree of b elements can be encoded using $2b = \frac{1}{2} \lg n$ bits

↪ # different Cartesian trees is $\leq 2^{2b} = 2^{\frac{1}{2} \lg n} = \left(2^{\lg n}\right)^{1/2} = \sqrt{n}$

↪ many equivalent blocks!

↪ **Exhaustive Tabulation Technique:**

1. represent each subproblem by storing its *type* (here: encoding of Cartesian tree)
2. *enumerate* all possible subproblem types and their solutions
3. use type as index in a large *lookup table*

Intrablock queries [2]

1. For each block, compute $2b$ bit representation of Cartesian tree
 - ▶ can be done in linear time
2. Compute large lookup table

Block type	i	j	RMQ(i, j)
⋮			
⋮			

▶ $\leq \sqrt{n}$ block types

▶ $\leq b^2$ combinations for i and j

↪ $\Theta(\sqrt{n} \cdot \log^2 n)$ rows

▶ each row can be computed in $O(\log n)$ time

↪ overall preprocessing: $O(n)$ time!

Discussion

▶ $\langle O(n), O(1) \rangle$ time solution for RMQ

\rightsquigarrow $\langle O(n), O(1) \rangle$ time solution for LCE in strings!

 optimal preprocessing and query time!

 a bit complicated

Research questions:

- ▶ Reduce the space usage
- ▶ Avoid access to A at query time