

9

Range-Minimum Queries

04 May 2021

Sebastian Wild

9 Range-Minimum Queries

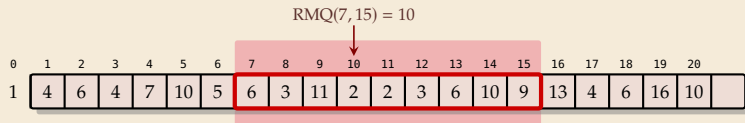
- 9.1 Introduction
- 9.2 RMQ, LCP, LCE, LCA — WTF?
- 9.3 Sparse Tables
- 9.4 Cartesian Trees
- 9.5 “Four Russians” Table

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
 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:

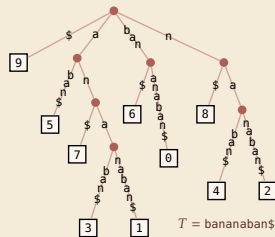
The *longest common extension (LCE)* data structure:

- ▶ **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} : $LCE(i, j) = \overset{\text{longest common prefix of } i\text{th and } j\text{th suffix}}{LCP(T_i, T_j)} \rightsquigarrow$ same thing, different name!
 $=$ string depth of
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!

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

Inverse suffix array: going left & right

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

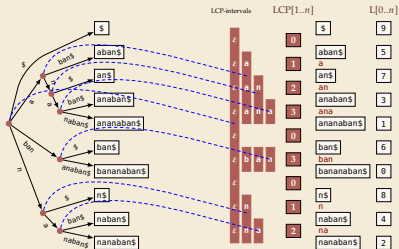
- ▶ $R[i] = r \iff L[r] = i$ $L = \text{leaf array}$
- \iff there are r suffixes that come before T_i in sorted order
- $\iff T_i$ has (0-based) *rank* $r \rightsquigarrow$ call $R[0..n]$ the *rank array*

i	$R[i]$	T_i	r	$L[r]$	$T_{L[r]}$
0	6 th	bananabans\$	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	ban\$	6	0	bananabans\$
7	2 th	ans\$	7	8	n\$
8	7 th	n\$	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!

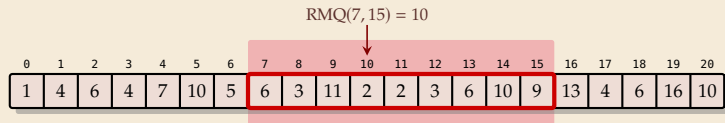
35

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 Sparse Tables

Trivial Solutions



- ▶ Two easy solutions show extreme ends of scale:

1. Scan on demand

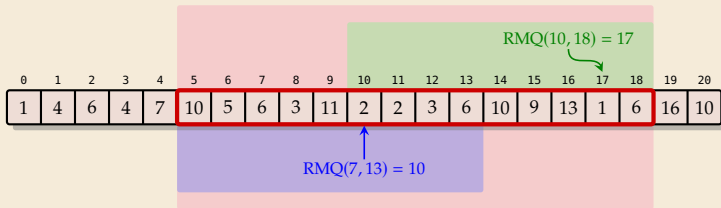
- ▶ no preprocessing at all
 - ▶ answer $\text{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min
- $\rightsquigarrow \langle O(1), O(n) \rangle$

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 .
 $\rightsquigarrow \leq n \cdot \lg n$ entries
- ▶ How to answer queries?

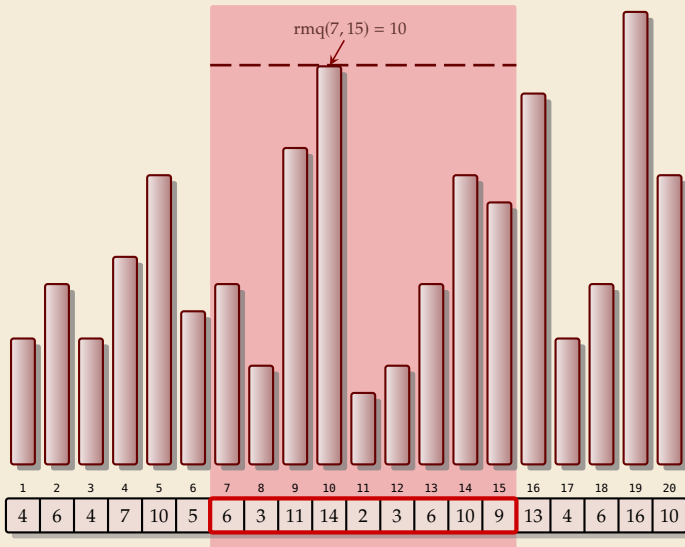


- ▶ Preprocessing can be done in $O(n \log n)$ times

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

9.4 Cartesian Trees

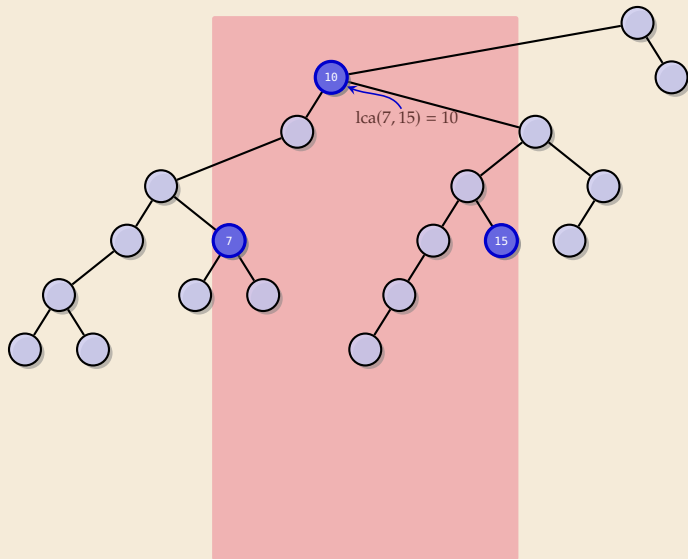
Range-maximum queries



- ▶ **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!

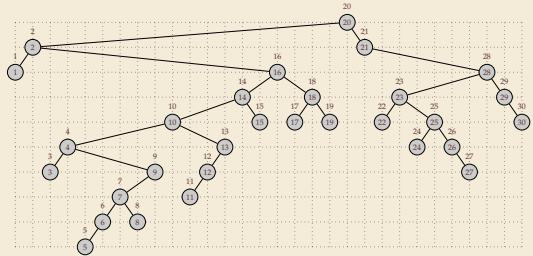
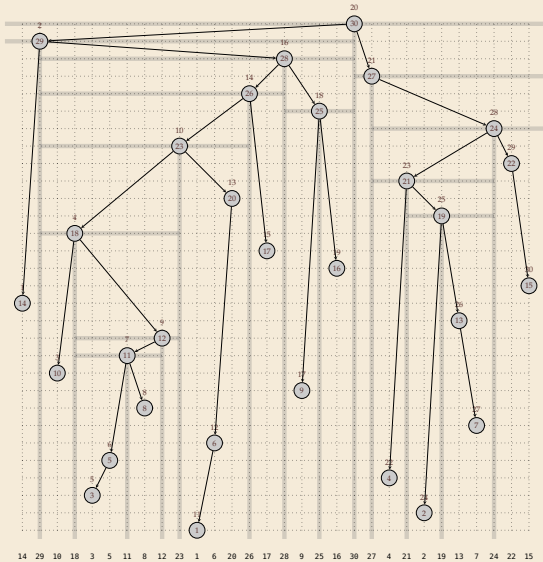
Range-maximum queries



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

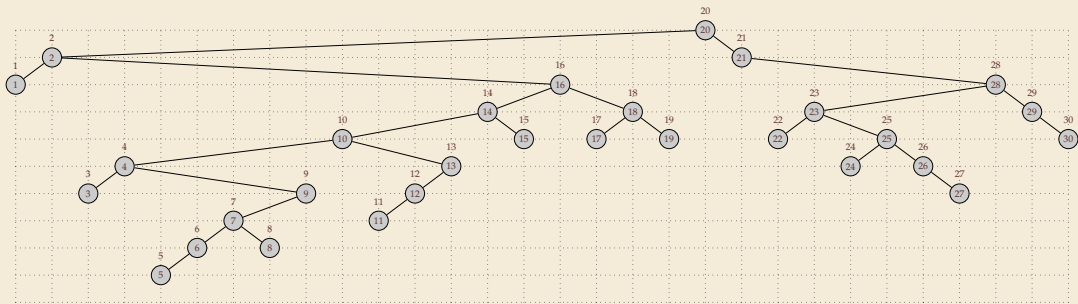
$$= \text{index of max}$$
- ▶ **Task:** Preprocess A ,
then answer RMQs fast
ideally constant time!
- ▶ **Cartesian tree:** (cf. *treap*)
construct binary tree by
sweeping line down
- ▶ $\text{rmq}(i, j) =$ inorder of
lowest common ancestor (LCA)
of i th and j th node in inorder

Cartesian Tree – Example



Counting binary trees

- ▶ all RMQ answers are determined by Cartesian tree
- ▶ How many different Cartesian trees are there for $A[0..n]$?
 - ▶ known result: Catalan numbers $\frac{1}{n+1} \binom{2n}{n}$
 - ▶ easy to see: $\leq 2^{2n}$



9.5 “Four Russians” Table

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 .


Query decomposition


Precomputing intra-block queries

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