9 Range-Minimum Queries

04 May 2021

Sebastian Wild

version `2021-05-05 10:38`

Outline

9 Range-Minimum Queries

9.1 Introduction

Range-minimum queries (RMQ)

array/numbers don't change

- **Given:** Static array $A[0..n)$ of numbers (any ordered objects)
- **Goal:** Find minimum in a range;
  A known in advance and can be preprocessed

$\mathrm{RMQ}(7, 15) = 10$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

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

Clicker Question

Given the array from the slides, what is $\mathrm{RMQ}_A(1,6)$ = 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

sli.do/comp526

Click on "Polls" tab

Rules of the Game

- comparison-based $\rightsquigarrow$ values don't matter, only relative order
- Two main quantities of interest:

  $\rightsquigarrow$ space usage $\le P(n)$

  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

Clicker Question

What do you think, what running times can we achieve? For a $\langle P(n), Q(n) \rangle$ time solution, enter “`<P(n),Q(n)>`”.

`sli.do/comp526`

Click on “Polls” tab

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:

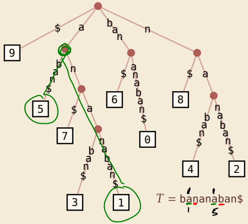
  The ***longest common extension (LCE)*** data structure:
  - **Given:** String $T[0..n-1]$
  - **Goal:** Answer LCE queries, i. e.,
    given positions i, j in T,
    how far can we read the same text from there?
    formally: $\mathrm{LCE}(i,j) = \max\{\ell : T[i..i+\ell) = T[j..j+\ell)\}$

$\rightsquigarrow$ use suffix tree of T!

- In $\mathcal{T}$: $\mathrm{LCE}(i,j) = \mathrm{LCP}(T_i, T_j)$ $\rightsquigarrow$ same thing, different name!
  $= $ string depth of *lowest common ancester (LCA)* of leaves $\boxed{i}$ and $\boxed{j}$

  longest common prefix of ith and jth suffix

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



15

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.

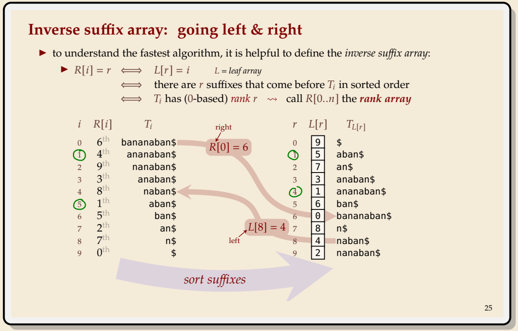
16

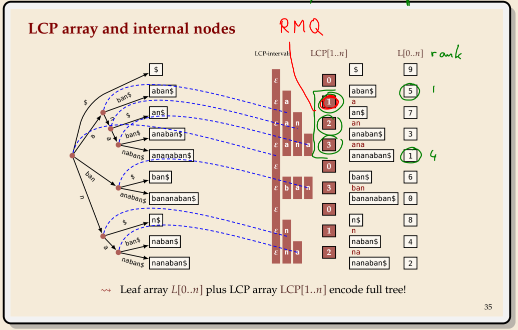
Finally: Longest common extensions

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

$$\boxed{\mathrm{LCE}(i,j) \;=\; \mathrm{LCP}\Big[\mathrm{RMQ}_{\mathrm{LCP}}\big(\min\{R[i],R[j]\}+1,\ \max\{R[i],R[j]\}\big)\Big]}$$

1 5
b a n a n a b a n $





RMQ Implications for LCE

- Recall: Can compute (inverse) suffix array and LCP array in $O(n)$ time

$\rightsquigarrow$ 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

$\Rightarrow$ really want $\langle O(n), O(1)\rangle$ solution
(best possible)

9.3 Sparse Tables

Trivial Solutions

$\mathrm{RMQ}(7, 15) = 10$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

- Two easy solutions show extreme ends of scale:

Trivial Solutions

$\mathrm{RMQ}(7, 15) = 10$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

- Two easy solutions show extreme ends of scale:

1. **Scan on demand**
   - no preprocessing at all
   - answer $\mathrm{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min

   $\rightsquigarrow$ $\langle O(1), O(n) \rangle$

Trivial Solutions

$\mathrm{RMQ}(7, 15) = 10$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

- Two easy solutions show extreme ends of scale:

1. **Scan on demand**

- no preprocessing at all
- answer $\mathrm{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min

$\rightsquigarrow$ $\langle O(1), O(n)\rangle$

2. **Precompute all** $0 \le i \le j < n$

- Precompute all answers in a big 2D array $M[0..n)[0..n)$
- queries simple: $\mathrm{RMQ}(i, j) = M[i][j]$

$\rightsquigarrow$ $\langle O(n^3), O(1)\rangle$

Trivial Solutions

RMQ(7, 15) = 10

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

- Two easy solutions show extreme ends of scale:

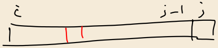
1. **Scan on demand**
   - no preprocessing at all
   - answer $\mathrm{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min

   $\rightsquigarrow \langle O(1), O(n) \rangle$

2. **Precompute all**
   - Precompute all answers in a big 2D array $M[0..n)[0..n)$
   - queries simple: $\mathrm{RMQ}(i, j) = M[i][j]$

   $\rightsquigarrow \langle O(n^3), O(1) \rangle$

   - Preprocessing can reuse partial results $\rightsquigarrow$ $\langle O(n^2), O(1) \rangle$



RMQ(i,j) = RMQ(i,j-1)
or j

Sparse Table

- **Idea:** Like "precompute-all", but keep only some entries
- store $M[i][j]$ iff $\ell = j - i + 1$ is 2^k.
  - $\leadsto$ $\leq n \cdot \lg n$ entries
  - $\leadsto$ Can be stored as $M'[i][k]$

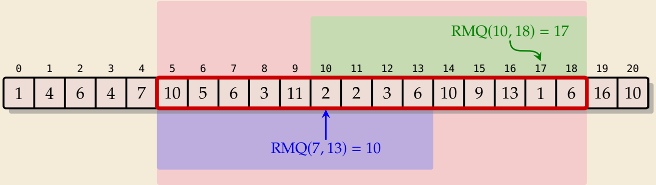
Sparse Table

- **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
  - $\rightsquigarrow$ Can be stored as $M'[i][k]$
- How to answer queries?

Sparse Table

- **Idea:** Like "precompute-all", but keep only some entries
- store $M[i][j]$ iff $\ell = j - i + 1$ is 2^k.
  - $\rightsquigarrow \le n \cdot \lg n$ entries
  - $\rightsquigarrow$ Can be stored as $M'[i][k]$

- How to answer queries?



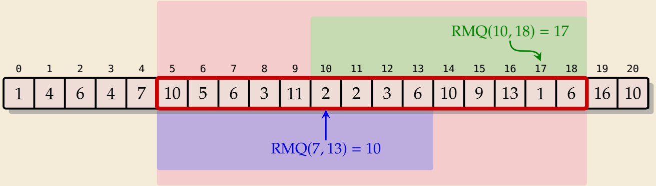
1. Find k with $\ell/2 \le 2^k \le \ell$
2. Cover range $[i..j]$ by
   2^k positions right from i and
   2^k positions left from j
3. $\text{RMQ}(i, j) =$
   $\arg\min\{A[rmq_1], A[rmq_2]\}$

   with $rmq_1 = \text{RMQ}(i, i + 2^k - 1)$
   $rmq_2 = \text{RMQ}(j - 2^k + 1, j)$

$= M'[i][k]$ $\qquad$ $= M'[j-2^k+1][k]$

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
  - $\rightsquigarrow$ Can be stored as $M'[i][k]$
- How to answer queries?



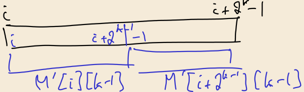
1. Find k with $\ell/2 \leq 2^k \leq \ell$
2. Cover range $[i..j]$ by
   2^k positions right from i and
   2^k positions left from j
3. $\text{RMQ}(i, j) =$
   $\arg\min\{A[rmq_1], A[rmq_2]\}$

   with $rmq_1 = \text{RMQ}(i, i + 2^k - 1)$
   $rmq_2 = \text{RMQ}(j - 2^k + 1, j)$

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

naive
$O(n \cdot n \log n) = O(n^2 \log n)$

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



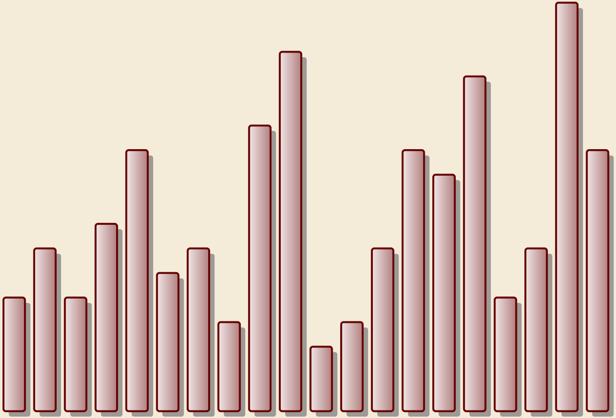
9.4 Cartesian Trees

RMQ & LCA

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
4	6	4	7	10	5	6	3	11	14	2	3	6	10	9	13	4	6	16	10

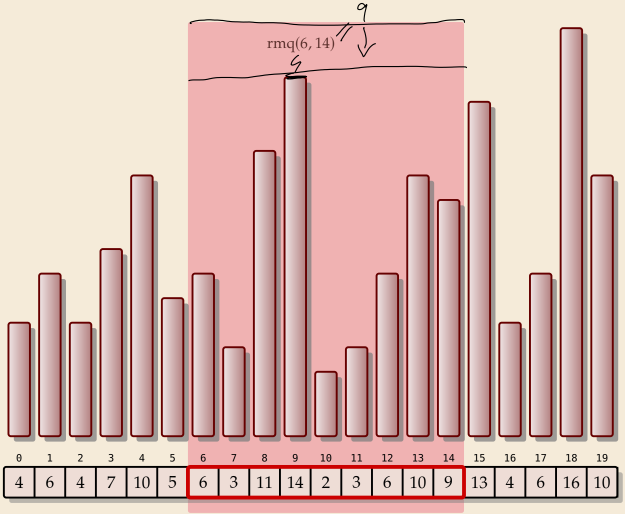
RMQ & LCA

⚠ RMQ = range-<u>max</u> query



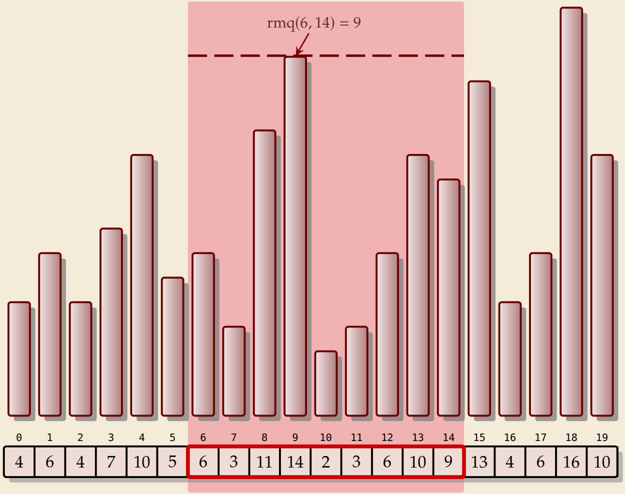
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
4	6	4	7	10	5	6	3	11	14	2	3	6	10	9	13	4	6	16	10

RMQ & LCA



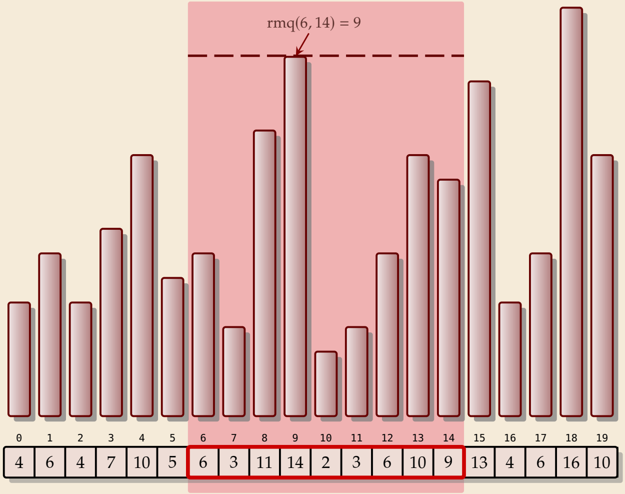
- **Range-max queries** on array A:
  $$\mathrm{rmq}_A(i,j) = \underset{i \le k \le j}{\arg\max}\, A[k]$$
  $$= \textit{index} \text{ of max}$$

RMQ & LCA



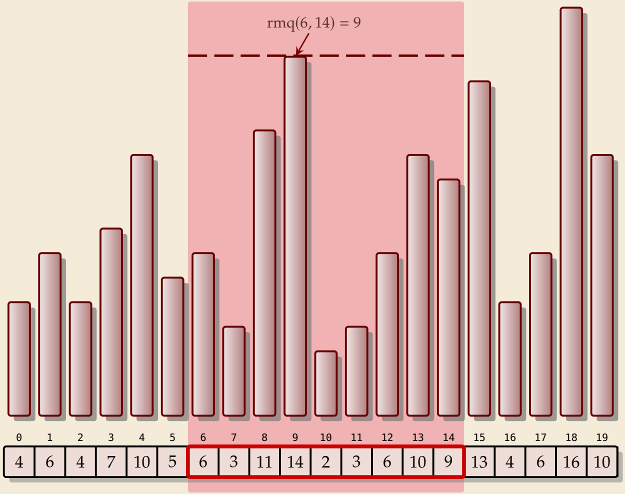
- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $=$ *index* of max

RMQ & LCA



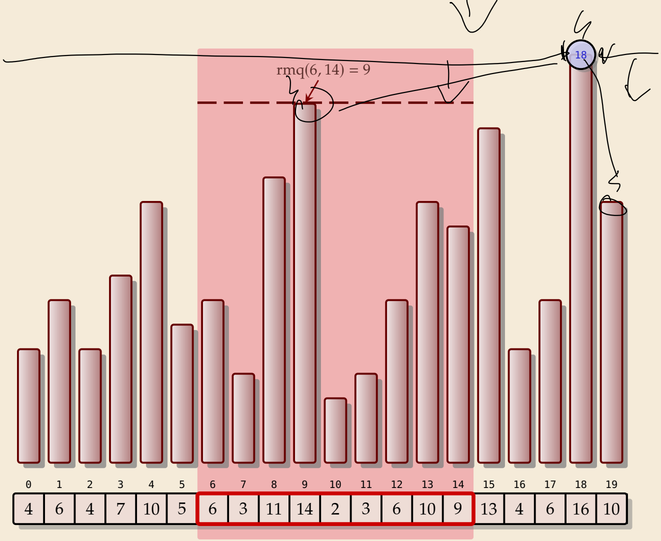
- **<u>R</u>ange-<u>m</u>ax <u>q</u>ueries** on array A:
  $$\mathrm{rmq}_A(i,j) = \underset{i \le k \le j}{\arg\max}\, A[k]$$
  $$= \textit{index} \text{ of max}$$
- **Task:** Preprocess A,
  then answer RMQs fast

RMQ & LCA



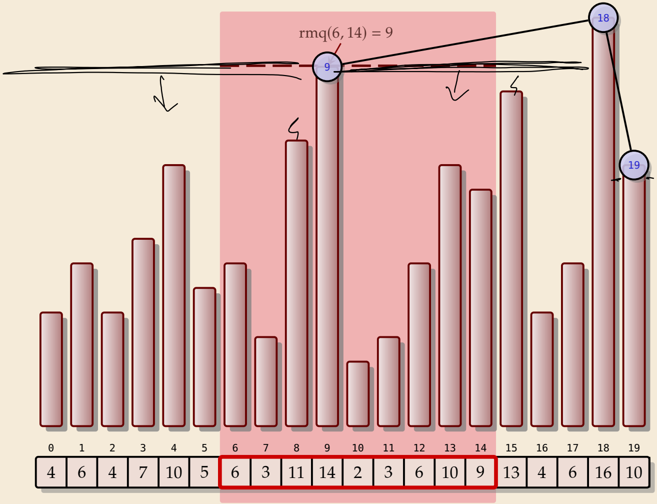
- **Range-max queries** on array A:
  $\text{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!

RMQ & LCA



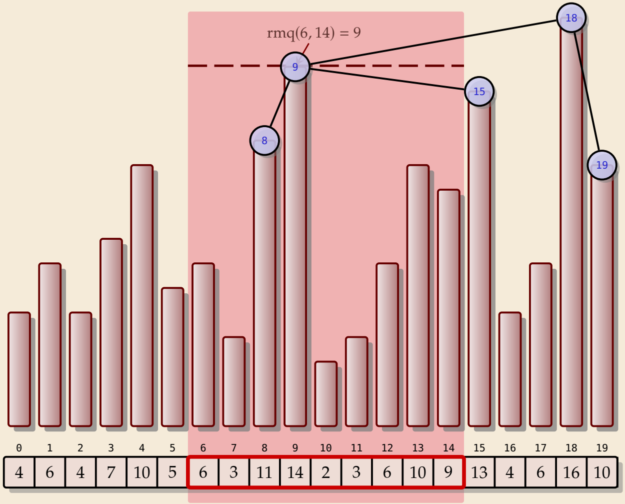
- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down

RMQ & LCA



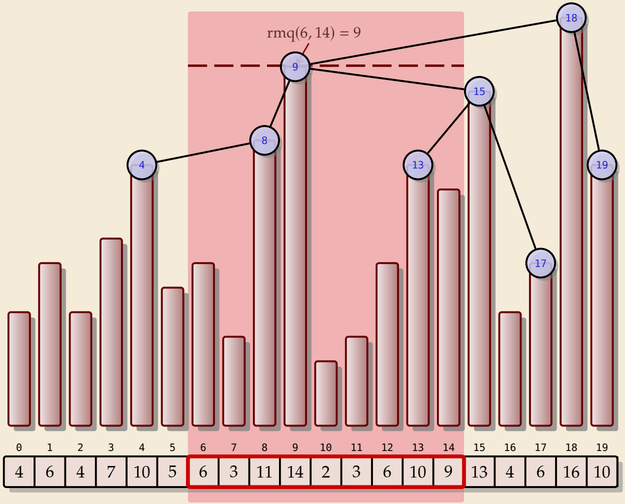
- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i,j) = \arg\max_{i \le k \le j} A[k]$
  $=$ *index* of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down

RMQ & LCA



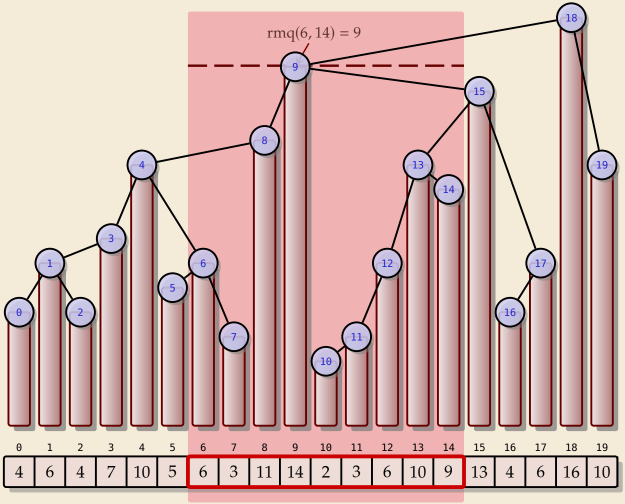
- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down

RMQ & LCA



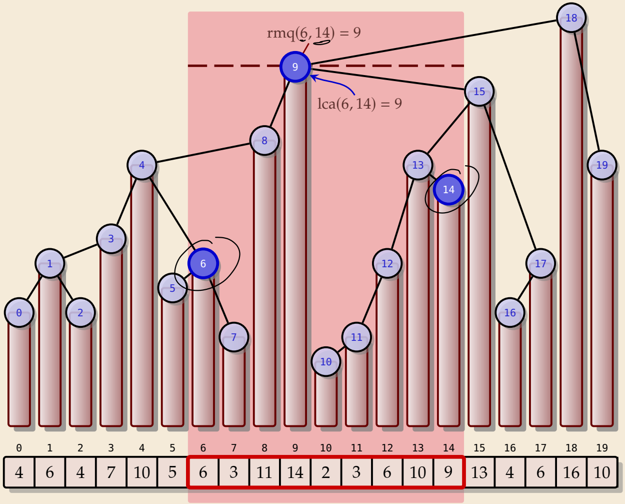
- **Range-max queries** on array A:
  $$\text{rmq}_A(i,j) = \arg\max_{i \le k \le j} A[k] = \textit{index of max}$$
- **Task:** Preprocess A, then answer RMQs fast ideally constant time!
- **Cartesian tree:** (cf. *treap*) construct binary tree by sweeping line down

RMQ & LCA



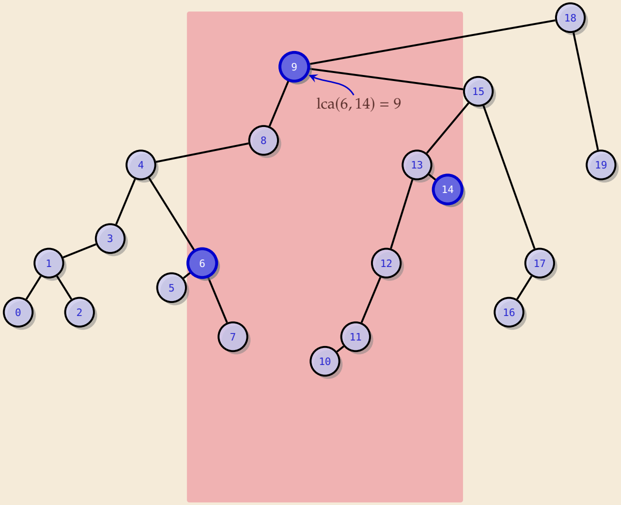
- **<u>R</u>ange-<u>m</u>ax <u>q</u>ueries** on array A:
  $$\mathrm{rmq}_A(i,j) = \arg\max_{i \le k \le j} A[k] = \textit{index} \text{ of max}$$
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down

RMQ & LCA



- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $= \mathit{index}$ of max
- **Task:** Preprocess A, then answer RMQs fast ideally constant time!
- **Cartesian tree:** (cf. *treap*) construct binary tree by sweeping line down
- $\mathrm{rmq}(i, j)$ = **lowest common ancestor** (LCA)

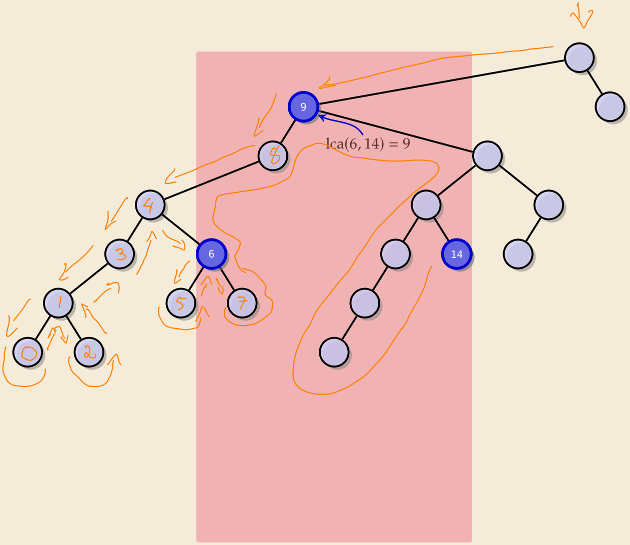
RMQ & LCA



- **R**ange-**m**ax **q**ueries on array A:
  $\mathrm{rmq}_A(i,j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down
- $\mathrm{rmq}(i,j)$ =
  lowest **c**ommon **a**ncestor (LCA)

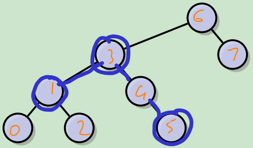
RMQ & LCA

inorder traversal



- **Range-max queries** on array A:
  $$\mathrm{rmq}_A(i,j) = \arg\max_{i \le k \le j} A[k]$$
  $$= \textit{index} \text{ of max}$$
- **Task:** Preprocess A, then answer RMQs fast ideally constant time!
- **Cartesian tree:** (cf. *treap*) construct binary tree by sweeping line down
- $\mathrm{rmq}(i,j)$ = inorder of **lowest common ancestor** (LCA) of ith and jth node in inorder

Clicker Question



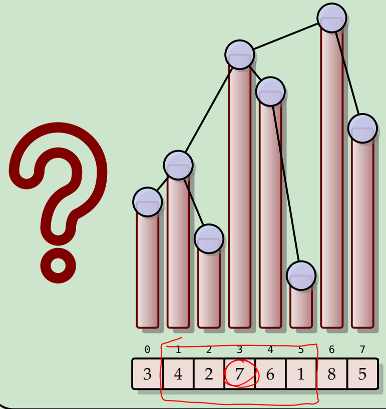
Given the (max-oriented) Cartesian tree for A on the left, what is $\mathrm{RMQ}_A(1, 5)$?

3

sli.do/comp526

Click on “Polls” tab

Clicker Question

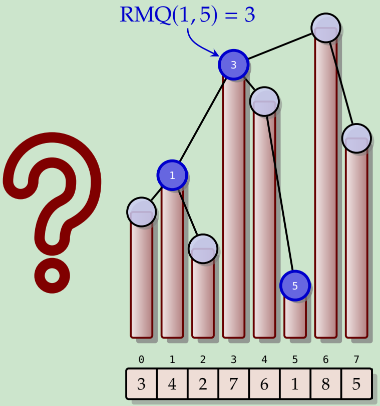


Given the (max-oriented) Cartesian tree for A on the left, what is $\mathrm{RMQ}_A(1,5)$?

sli.do/comp526

Click on "Polls" tab

Clicker Question

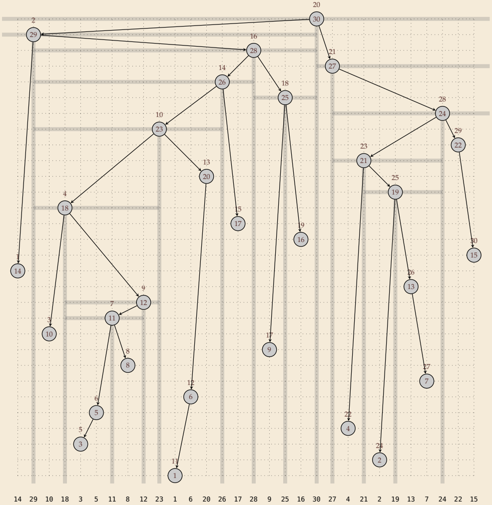


Given the (max-oriented) Cartesian tree for A on the left, what is $\mathrm{RMQ}_A(1, 5)$?

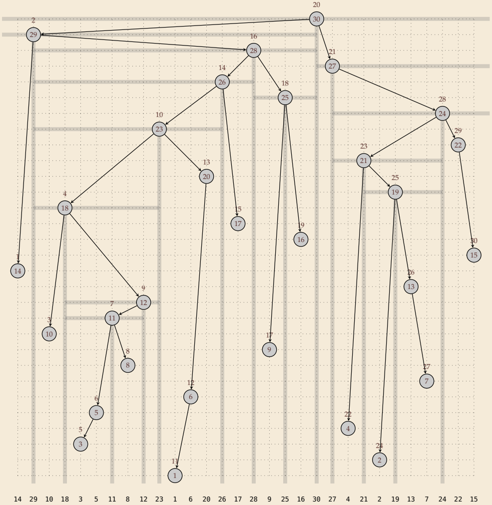
`sli.do/comp526`

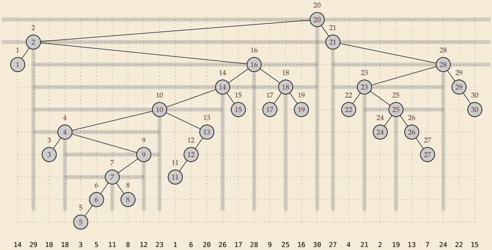
Click on "Polls" tab

Cartesian Tree – Larger Example

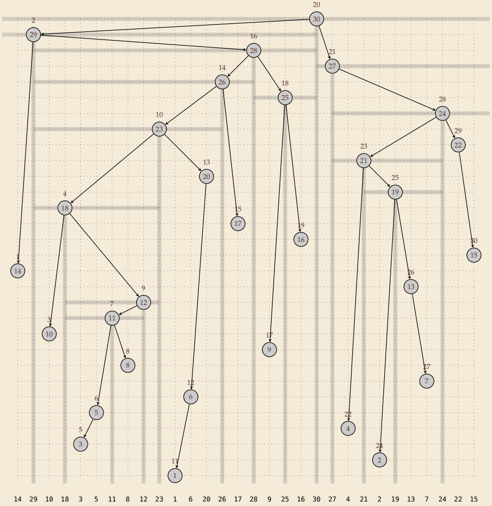


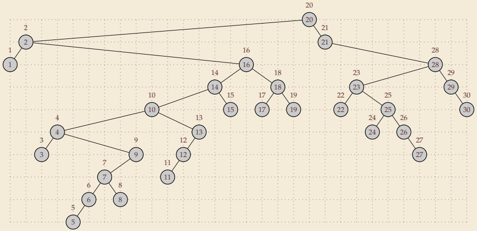
Cartesian Tree – Larger Example



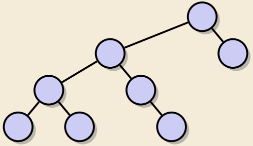


Cartesian Tree – Larger Example





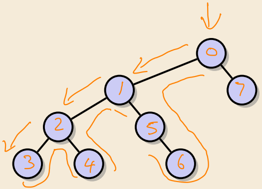
Counting binary trees



- Given the Cartesian tree,
  all RMQ answers are determined

  and vice versa!

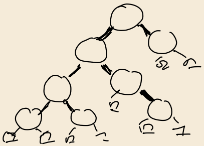
Counting binary trees



1111111 00 000100000
0 1 2 3

- Given the Cartesian tree,
  all RMQ answers are determined

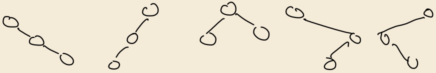
  and vice versa!



- 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: $\leq 2^{2n}$

$n = 3$

$\Rightarrow$ 5 trees



$\leadsto$ many arrays will give rise to the same Cartesian tree

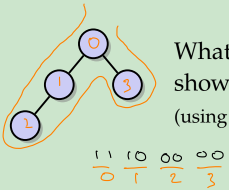
Can we exploit that?

code: in a preorder traversal
encode each node

$\boxed{1 \mid 0 \mid 1 \mid 1}$ but a right child

no left child

Clicker Question



What binary string corresponds to the tree shown on the left?

(using the encoding just discussed)

`sli.do/comp526`

Click on “Polls” tab

9.5 “Four Russians” Table

All Russian?

- What will follow is an algorithmic technique 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 clear if all are Russians!

  (Arlazarov and Kronrod are Russian)

All Russian?

- What will follow is an algorithmic technique 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 clear if all are Russians!

  (Arlazarov and Kronrod are Russian)

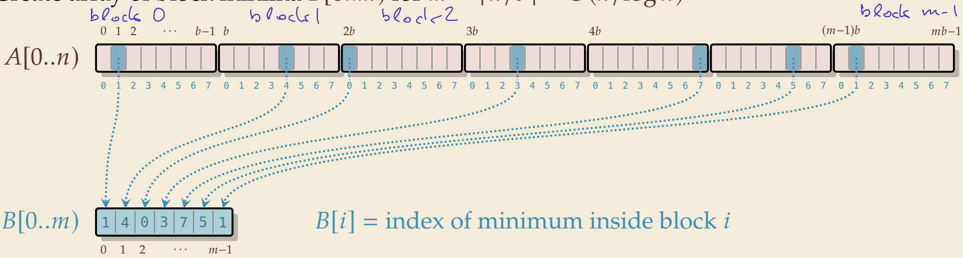
- American authors coined the slightly derogatory “Method of Four Russians” . . . name now in wide use

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 $m = \lceil n/b \rceil = O(n/\log n)$

$b = 8$



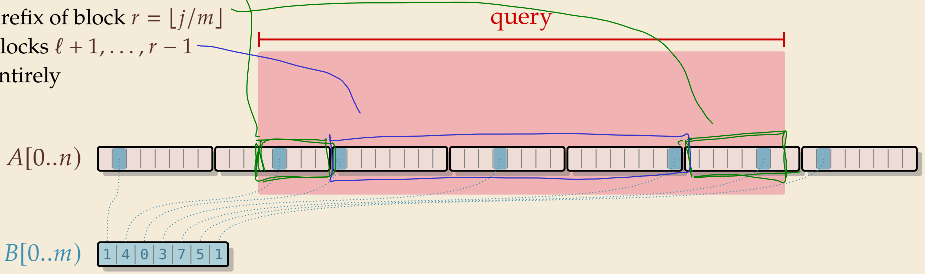
$B[i]$ = index of minimum inside block i

$\rightsquigarrow$ Use sparse tables for B.

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

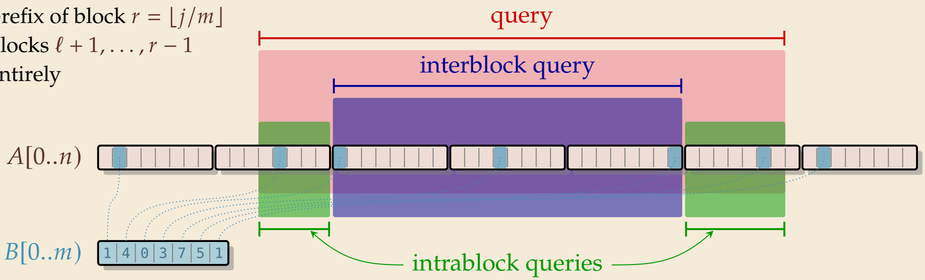
Query decomposition

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



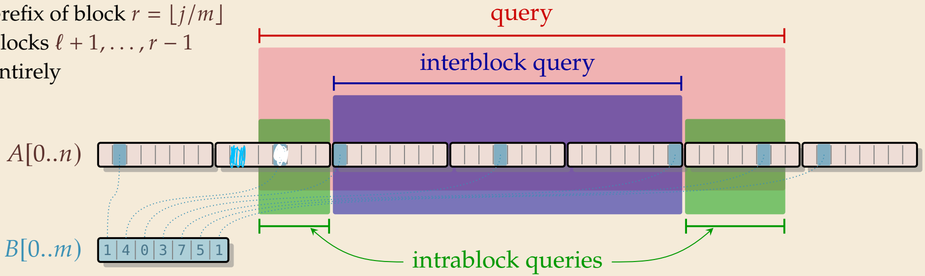
Query decomposition

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



Query decomposition

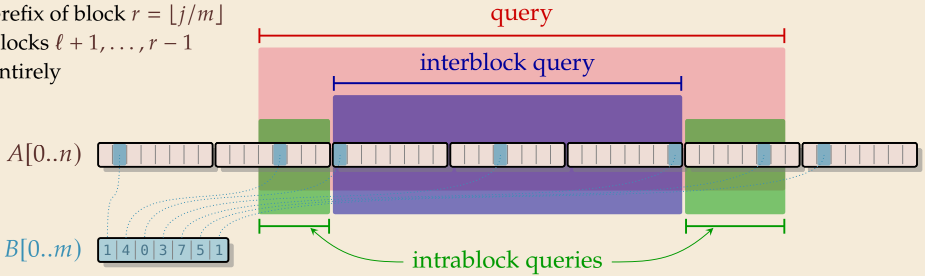
- Query $\mathrm{RMQ}_A(i, j)$ covers
  - suffix of block $\ell = \lfloor i/m \rfloor$
  - prefix of block $r = \lfloor j/m \rfloor$
  - blocks $\ell + 1, \ldots, r - 1$ entirely



- $\mathrm{RMQ}_A(i, j) \;=\; \underset{k \in K}{\arg\min}\, A[k]$ with $K = \left\{ \begin{array}{l} \mathrm{RMQ}_{\text{block } \ell}\big(i - \ell b,\ (\ell+1)b - 1\big), \\ \underbrace{b \cdot \mathrm{RMQ}_B(\ell + 1, r - 1)}_{\text{block}} + \\ \qquad \underbrace{B\big[\mathrm{RMQ}_B(\ell + 1, r - 1)\big]}_{\text{block-local offset}}, \\ \mathrm{RMQ}_{\text{block } r}\big(rb,\ j - rb\big) \end{array} \right\}$

Query decomposition

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

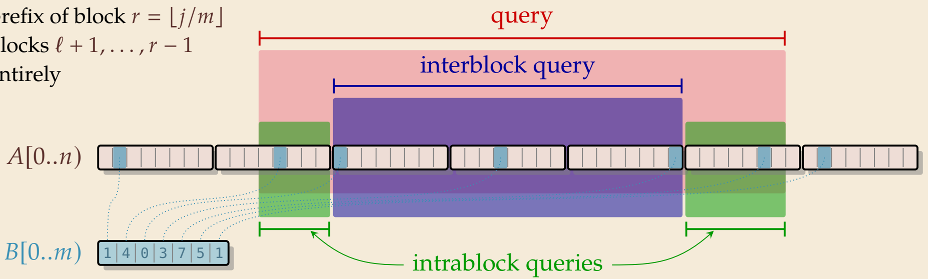


- $\mathrm{RMQ}_A(i,j) = \underset{k \in K}{\arg\min}\, A[k]$ with $K = \left\{ \begin{array}{l} \mathrm{RMQ}_{\text{block } \ell}\big(i - \ell b,\ (\ell+1)b - 1\big), \\ b \cdot \mathrm{RMQ}_B(\ell+1, r-1) + \\ \qquad B\big[\mathrm{RMQ}_B(\ell+1, r-1)\big], \\ \mathrm{RMQ}_{\text{block } r}\big(rb,\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

$\rightsquigarrow$ only 3 possible values to check if intrablock and interblock queries known

Query decomposition

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



- $\mathrm{RMQ}_A(i, j) \;=\; \underset{k \in K}{\arg\min}\, A[k] \quad \text{with } K = \left\{ \begin{array}{l} \mathrm{RMQ}_{\text{block } \ell}\big(i - \ell b,\, (\ell+1)b - 1\big), \\ b \cdot \mathrm{RMQ}_B(\ell+1, r-1) + \\ \qquad B\big[\mathrm{RMQ}_B(\ell+1, r-1)\big], \\ \mathrm{RMQ}_{\text{block } r}\big(rb,\, j - rb\big) \end{array} \right\}$

$\rightsquigarrow$ only 3 possible values to check
if intrablock and interblock queries known ✓

Intrablock queries [1]

$\rightsquigarrow$ 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!

Intrablock queries [1]

$\rightsquigarrow$ 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

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

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

  $\rightsquigarrow$ many equivalent blocks!

Intrablock queries [1]

$\rightsquigarrow$ 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

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

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

  $\rightsquigarrow$ many equivalent blocks!

$\rightsquigarrow$ ***"Four Russians" Technique:***

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

Intrablock queries [2]

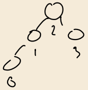
1. For each block, compute $2b$ bit representation of Cartesian tree
   - can be done in linear time

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

$b = 4$



Block type	i	j	RMQ(i, j)
$\vdots$			
11100000	0	1	1
	0	2	2
	0	3	2
	1	2	2
	1	3	2
	2	3	2
$\vdots$			

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	$\mathrm{RMQ}(i,j)$
$\vdots$			
$\vdots$			

- $\leq \sqrt{n}$ block types
- $\leq b^2$ combinations for i and j

$\rightsquigarrow$ $\Theta\left(\sqrt{n} \cdot \log^2 n\right)$ rows

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

$\rightsquigarrow$ 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!

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

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