14 Range-Minimum Queries

10 February 2025

Prof. Dr. Sebastian Wild

CS566 (Wintersemester 2024/25)
Philipps-Universität Marburg

version `2025-01-16 11:52` *H*

Learning Outcomes

Unit 14: ***Range-Minimum Queries***

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.

Outline

14 Range-Minimum Queries

14.1 Introduction

14.2 RMQ, LCP, LCE, LCA — WTF?

14.3 Trivial Solutions & Sparse Tables

14.4 Cartesian Trees

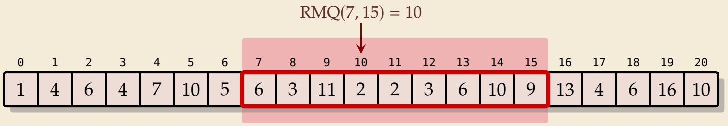
14.5 Exhaustive Tabulation

14.1 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:

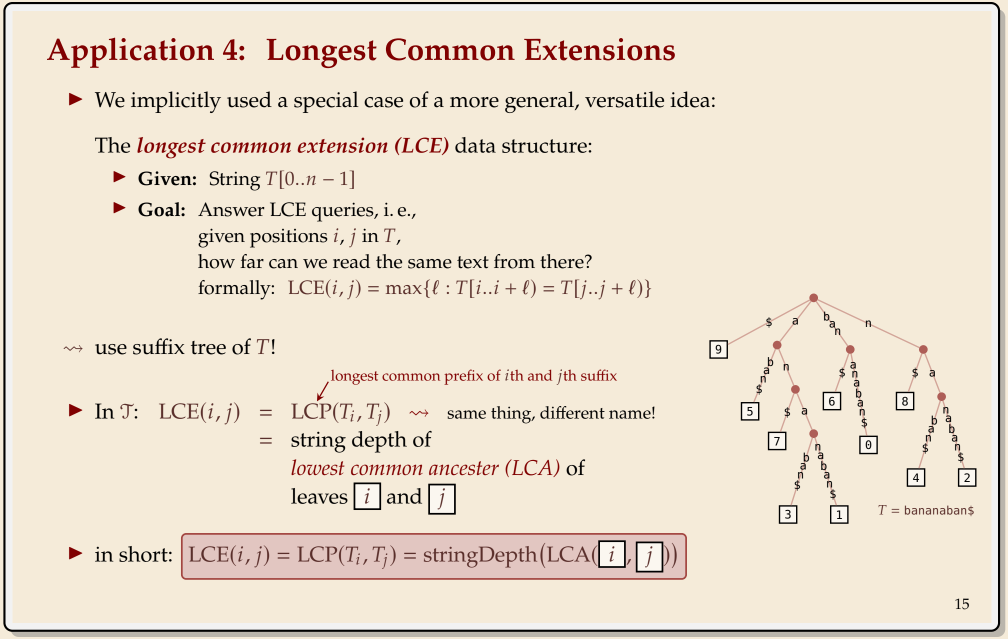
  $\rightsquigarrow$ space usage $\leq P(n)$

  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

14.2 RMQ, LCP, LCE, LCA — WTF?

Recall Unit 13



Application 4: Longest Common Extensions

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

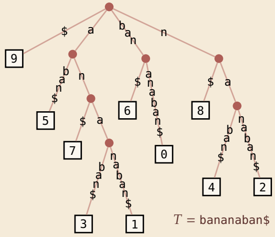
  The *longest common extension (LCE)* data structure:

  - **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: $\text{LCE}(i,j) = \max\{\ell : T[i..i+\ell) = T[j..j+\ell)\}$

$\rightsquigarrow$ use suffix tree of T!

longest common prefix of ith and jth suffix

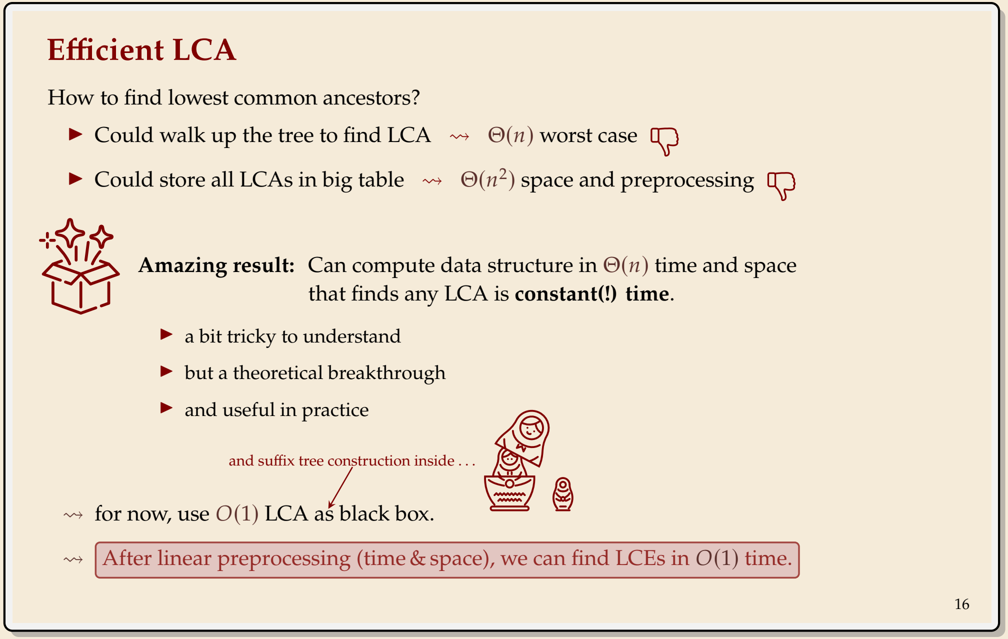
- In $\mathcal{T}$: $\text{LCE}(i,j) = \text{LCP}(T_i, T_j)$ $\rightsquigarrow$ same thing, different name!
  $=$ string depth of
  lowest common ancester (LCA) of
  leaves $\boxed{i}$ and $\boxed{j}$



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

15

Recall Unit 13



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.

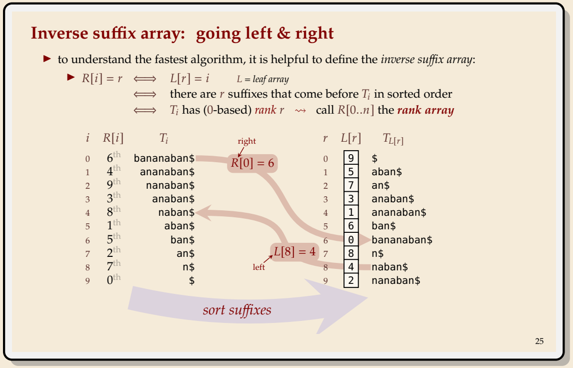
$\rightsquigarrow$ After linear preprocessing (time & space), we can find LCEs in $O(1)$ time.

16

Finally: Longest common extensions

- In Unit 13: Left question open how to compute LCA in suffix trees
- But: Enhanced Suffix Array makes life easier!

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



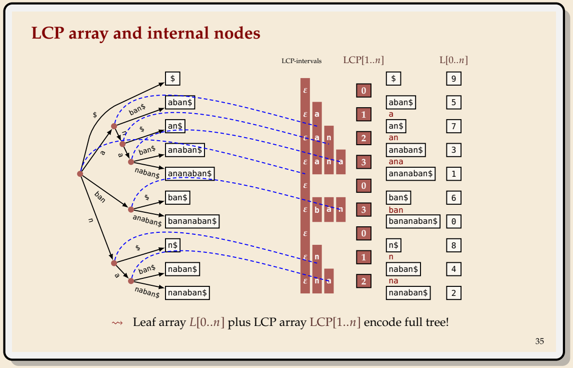
Inverse suffix array: going left & right

- to understand the fastest algorithm, it is helpful to define the *inverse suffix array*:
  - $R[i] = r \iff L[r] = i$ *L = leaf array*
    $\iff$ there are r suffixes that come before T_i in sorted order
    $\iff$ T_i has (0-based) *rank* r $\leadsto$ call $R[0..n]$ the ***rank array***

i	$R[i]$	T_i
0	6th	bananaban$
1	4th	ananaban$
2	9th	nanaban$
3	3th	anaban$
4	8th	naban$
5	1th	aban$
6	5th	ban$
7	2th	an$
8	7th	n$
9	0th	$

r	$L[r]$	$T_{L[r]}$
0	9	$
1	5	aban$
2	7	an$
3	3	anaban$
4	1	ananaban$
5	6	ban$
6	0	bananaban$
7	8	n$
8	4	naban$
9	2	nanaban$

25



LCP array and internal nodes

	LCP[1..n]	L[0..n]
$		9
	0	
aban$		5
a	1	
an$		7
an	2	
anaban$		3
ana	3	
ananaban$		1
	0	
ban$		6
ban	3	
bananaban$		0
	0	
n$		8
n	1	
naban$		4
na	2	
nanaban$		2

$\leadsto$ Leaf array $L[0..n]$ plus LCP array $\mathrm{LCP}[1..n]$ encode full tree!

35

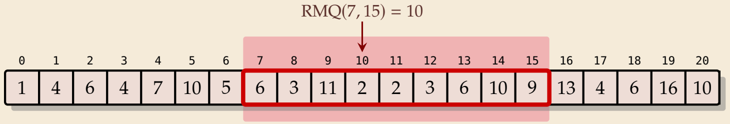
RMQ Implications for LCE

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

$\leadsto$ 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

14.3 Trivial Solutions & Sparse Tables

Trivial Solutions



- 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**

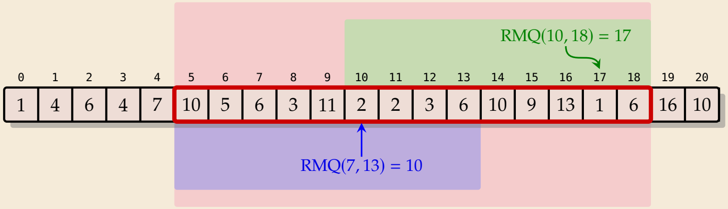
  - Precompute all answers in a big 2D array $M[0..n)[0..n)$
  - queries simple: $\mathrm{RMQ}(i, j) = M[i][j]$

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

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

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. $\mathrm{RMQ}(i,j) =$
   $\arg\min\{A[rmq_1], A[rmq_2]\}$

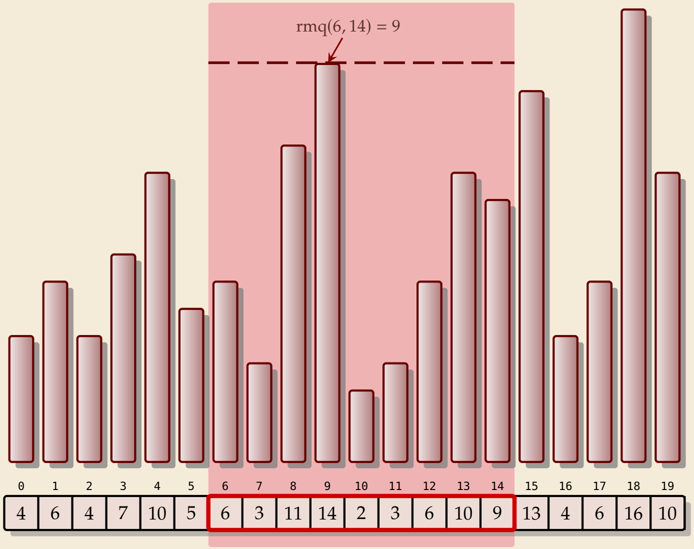
   with $rmq_1 = \mathrm{RMQ}(i, i + 2^k - 1)$
   $rmq_2 = \mathrm{RMQ}(j - 2^k + 1, j)$

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

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

14.4 Cartesian Trees

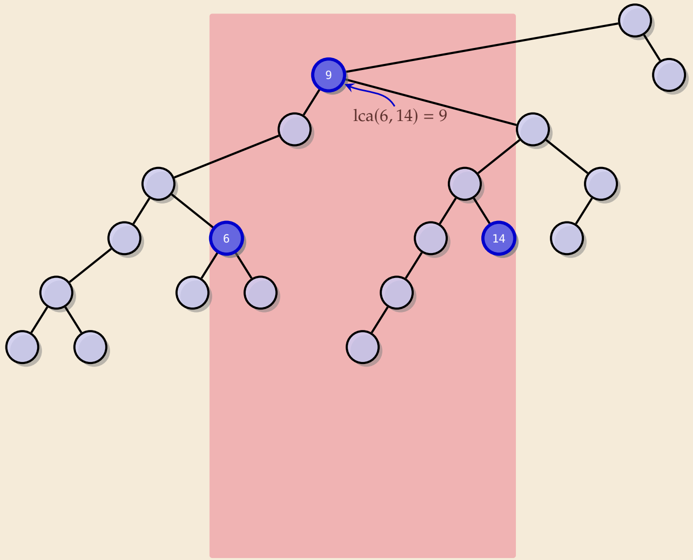
RMQ & LCA



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

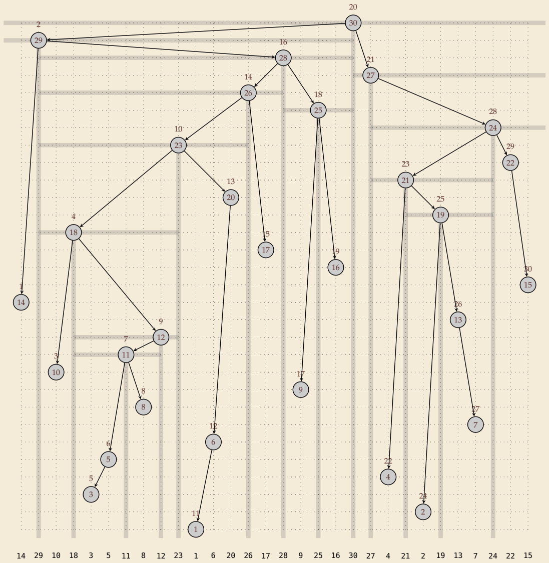
- **Task:** Preprocess A, then answer RMQs fast ideally constant time!

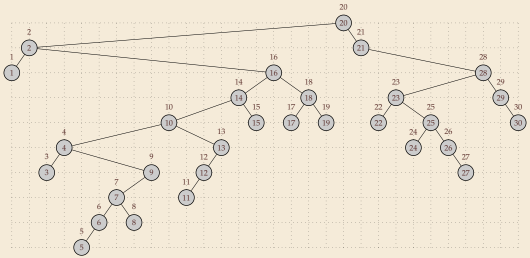
RMQ & LCA



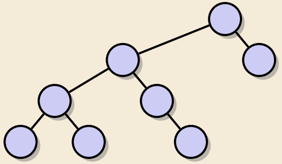
- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i,j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down
- $\mathrm{rmq}(i,j)$ = inorder of
  lowest common ancestor (LCA)
  of ith and jth 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: $\le 2^{2n}$

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

Can we exploit that?

14.5 Exhaustive Tabulation

Four Russians?

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

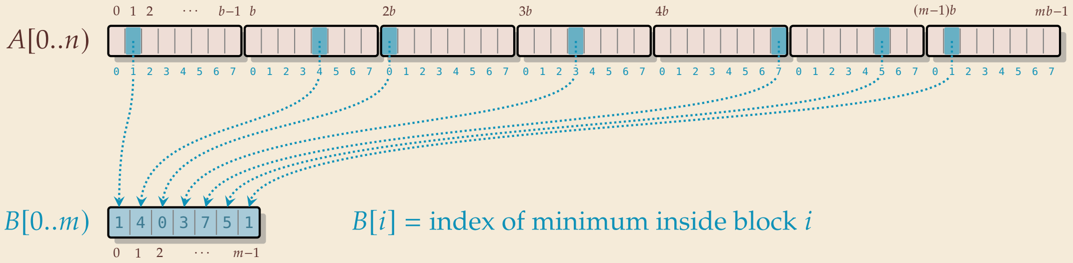
- The algorithmic technique was published 1970 by
  V. L. Arlazarov, E. A. Dinitz, M. A. Kronrod, and I. A. Faradžev
- all worked in Moscow at that time . . . but not even clear if all are Russians!

  (Arlazarov and Kronrod are Russian)
- American authors coined the othering term "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 $m = \lceil n/b \rceil = O(n/\log n)$

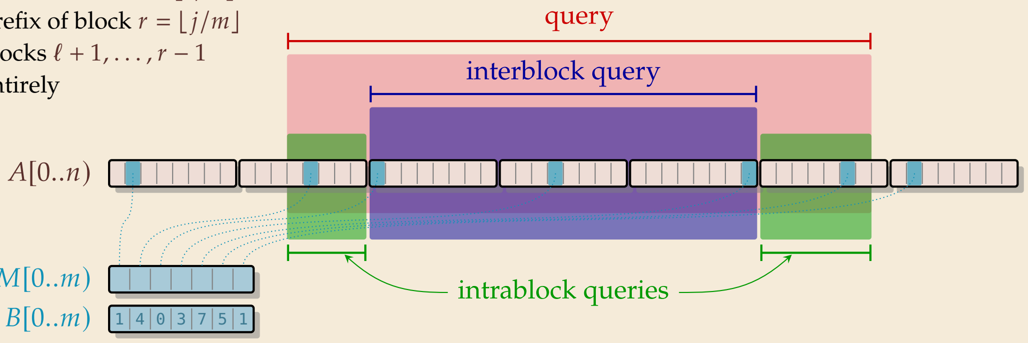


$\rightsquigarrow$ Use sparse tables for B.

$\rightsquigarrow$ 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



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

$\rightsquigarrow$ 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

  $\nwarrow$ 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!

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

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

$\rightsquigarrow$ $\Theta(\sqrt{n} \cdot \log^2 n)$ 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!

👍 optimal preprocessing and query time!

👎 a bit complicated

Research questions:

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