

ALGORITHMICS\$APPLIED  
APPLIEDALGORITHMICS\$  
CS\$APPLIEDALGORITHMICS\$  
DALGORITHMICS\$APPLIE  
EDALGORITHMICS\$APPLI  
GORITHMICS\$APPLIEDAL  
HMICS\$APPLIEDALGORIT  
ICS\$APPLIEDALGORITHM  
IEDALGORITHMICS\$APPL  
ITMICS\$APPLIEDALGOR  
LQTHMICS\$APPLIE  
LIEDALGORITHMICS\$APP  
MICS\$APPLIEDALGO  
ORITHMICS\$APPLIEDALG  
PPLIEDALGORITHMICS\$A  
PPLIEDALGORITHMICS\$A  
RITHMICS\$APPLIEDALGO  
S\$APPLIEDALGORITHMIC  
THMICS\$APPLIEDALGORI

# 9 Range-Minimum Queries

04 May 2021

Sebastian Wild

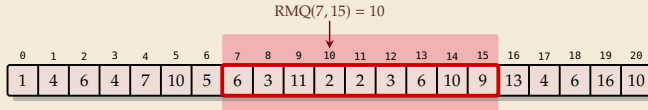
## 9 Range-Minimum Queries

- 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)

- ▶ **Given:** Static array  $A[0..n)$  of numbers (any ordered objects)  
*array/numbers don't change*
- ▶ **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

## Clicker Question



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

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](https://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:
  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

## Clicker Question



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

[sli.do/comp526](https://sli.do/comp526)

Click on “Polls” tab

## 9.2 RMQ, LCP, LCE, LCA — WTF?



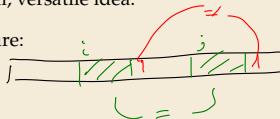
# Recall Unit 6

## Application 4: Longest Common Extensions

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

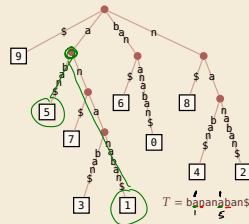
The *longest common extension (LCE)* data structure:

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



→ use suffix tree of  $T$ !

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





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

# 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*:

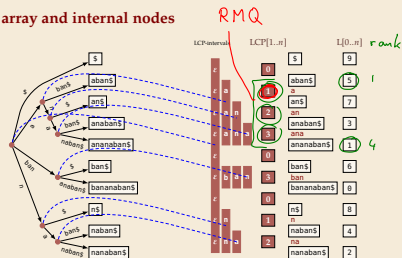
- $R[i] = r \iff L[r] = i$   $L$  = leaf array
- $\iff$  there are  $r$  suffixes that come before  $T_i$  in sorted order
- $\iff T_i$  has (0-based) *rank*  $r \rightsquigarrow$  call  $R[0..n]$  the *rank array*

$i$	$R[i]$	$T_i$	$r$	$L[r]$	$T_{L[r]}$
0	6 <sup>th</sup>	bananaban\$	9	\$	
1	4 <sup>th</sup>	ananaban\$	5	abans\$	
2	9 <sup>th</sup>	nanaban\$	7	ans\$	
3	3 <sup>th</sup>	anaban\$	3	anaban\$	
4	8 <sup>th</sup>	naban\$	1	ananaban\$	
5	1 <sup>th</sup>	aban\$	6	ban\$	
6	5 <sup>th</sup>	ban\$	0	bananaban\$	
7	2 <sup>th</sup>	an\$	8	n\$	
8	7 <sup>th</sup>	n\$	4	naban\$	
9	0 <sup>th</sup>	\$	2	nanaban\$	

sort suffixes

25

## LCP array and internal nodes



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

35

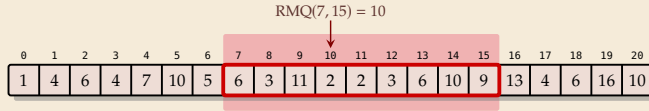
## RMQ Implications for LCE

- 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

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

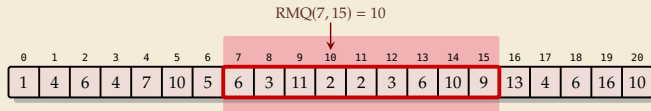
## 9.3 Sparse Tables

# Trivial Solutions



- Two easy solutions show extreme ends of scale:

# Trivial Solutions

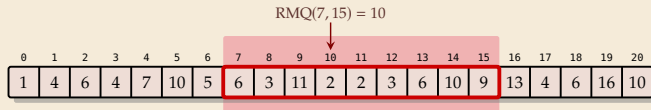


- ▶ Two easy solutions show extreme ends of scale:

## 1. Scan on demand

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

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

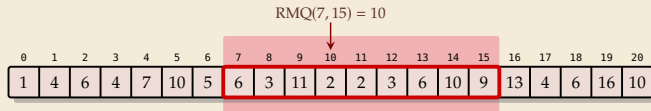
## 2. Precompute all

$$0 \leq i \leq j < n$$

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



# Trivial Solutions



- ▶ Two easy solutions show extreme ends of scale:

## 1. Scan on demand

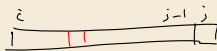
- ▶ no preprocessing at all
  - ▶ answer  $\text{RMQ}(i, j)$  by scanning through  $A[i..j]$ , keeping track of min
- $\rightsquigarrow \langle O(1), O(n) \rangle$

## 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  $\rightsquigarrow \langle O(n^2), O(1) \rangle$



$$\text{RMQ}(i, j) = \text{RMQ}(i, j-1) \text{ or } j$$