7 Googling Genomes

18 December 2025

Prof. Dr. Sebastian Wild

CS594 (Winter 2025-26)
Philipps-Universität Marburg
version `2026-01-15 13:11`

Outline

7 Googling Genomes

Recall Unit 6

Application 4: Longest Common Extensions

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

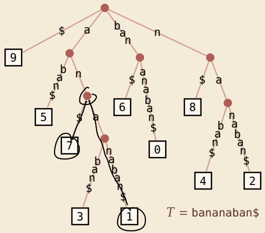
  The ***longest common extension (LCE)*** data structure:
  - **Given:** String $T[0..n)$
  - **Goal:** Answer LCE queries, i. e.,
    given positions i, j in T,
    how far can we read the same text from there?
    formally: $\mathrm{LCE}(i, j) = \max\{\ell : T[i..i+\ell) = T[j..j+\ell)\}$

$\rightsquigarrow$ use suffix tree of T!

(length of) longest common prefix
of ith and jth suffix

- 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 $\boxed{j}$

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



17

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 in **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.

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

7.1 Range-Minimum Queries

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

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

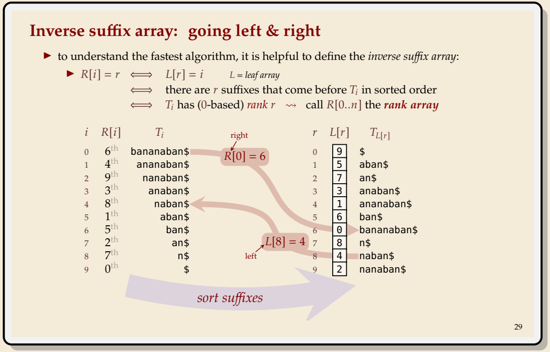
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

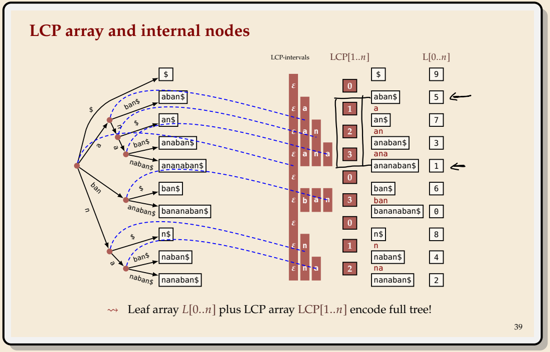
- **Nitpicks:**
  - Report *index* of minimum, not its value
  - Report *leftmost* position in case of ties

Finally: Longest common extensions

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

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





Rules of the Game

- For the following, consider RMQ on arbitrary arrays
- 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

Rules of the Game

- For the following, consider RMQ on arbitrary arrays
- 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

RMQ Implications for LCE

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

$\rightsquigarrow$ $\langle P(n), Q(n) \rangle$ time RMQ data structure implies
$\langle P(n) + O(n), Q(n) \rangle$ time LCE data structure

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:

Trivial Solutions

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

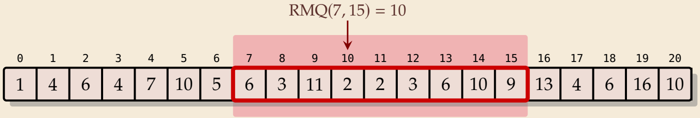
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

- 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



- 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: $\mathrm{RMQ}(i, j) = M[i][j]$

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

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

7.2 RMQ – Sparse Table Solution

Sparse Table

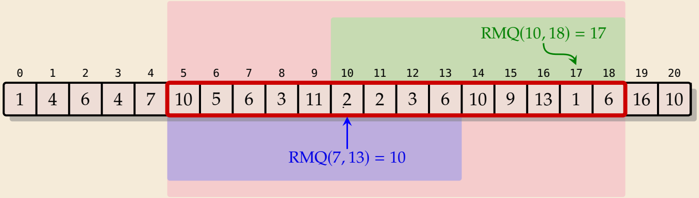
- **Idea:** Like "precompute-all", but keep only *some* entries
- store $M[i][j]$ iff $\ell = j - i + 1$ is 2^k.
  - $\rightsquigarrow$ $\le n \cdot \lg n$ entries
  - $\rightsquigarrow$ Can be stored as $M'[i][k]$

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?

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?

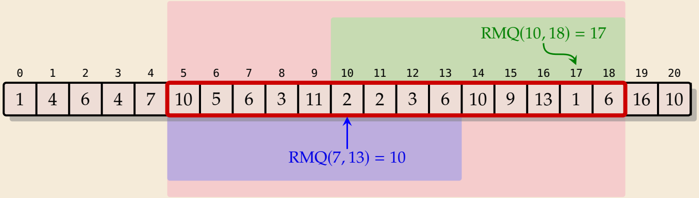


1. Find k with $\ell/2 \le 2^k \le \ell$
2. Cover range $[i..j]$ by
   2^k positions right from i and
   2^k positions left from j
3. $\mathrm{RMQ}(i,j) =$
   $\arg\min\{A[rmq_1], A[rmq_2]\}$

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

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!

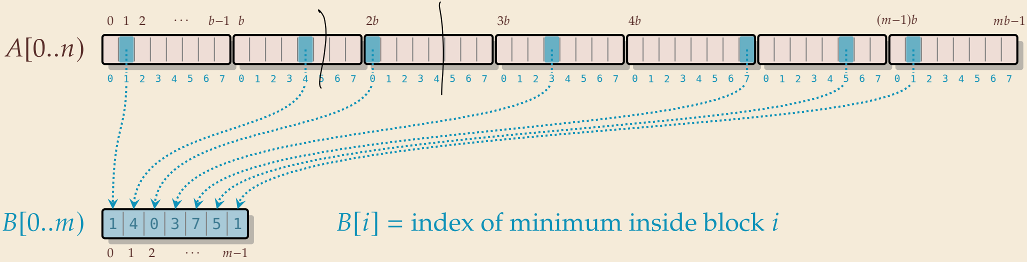
Bootstrapping

- We know a $\langle O(n \log n), O(1) \rangle$ time solution
- If we use that for $m = \Theta(n/\log n)$ elements, $O(m \log m) = O(n)$!

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, $O(m \log m) = O(n)$!

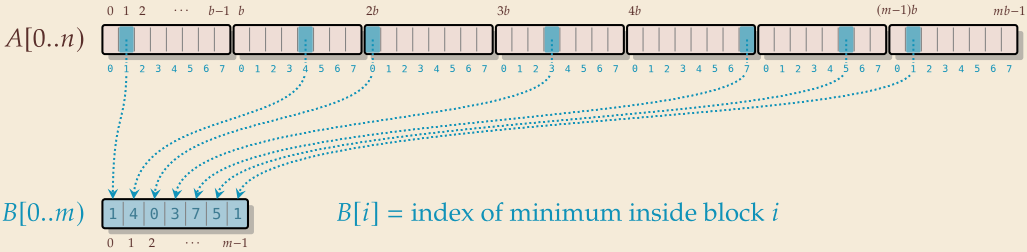
- Break A into blocks of $b = O(\log n)$ numbers
- Create array of block minima $B[0..m)$ for $m = \lceil n/b \rceil = O(n/\log n)$



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

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 $b = O(\log n)$ numbers
- Create array of block minima $B[0..m)$ for $m = \lceil n/b \rceil = O(n/\log n)$

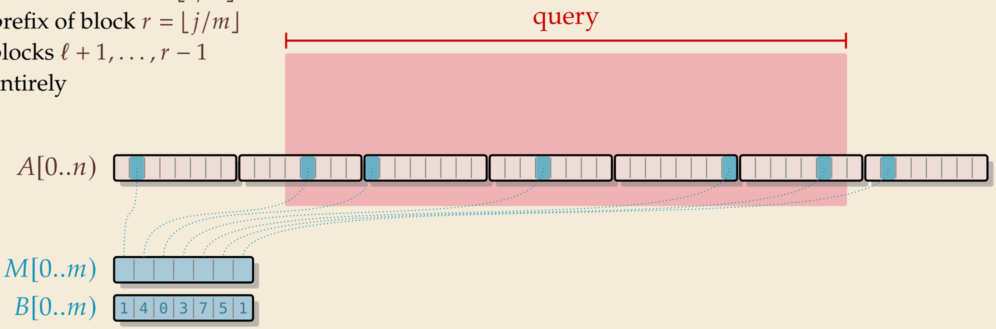


$\rightsquigarrow$ Use sparse table solution for B.

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

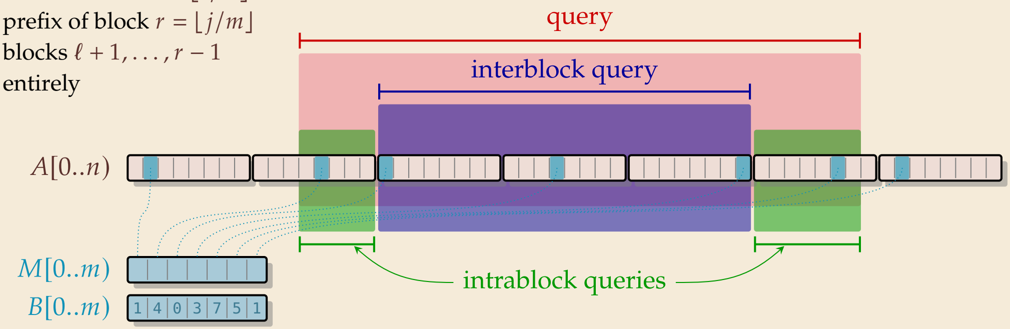
Query decomposition

- 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



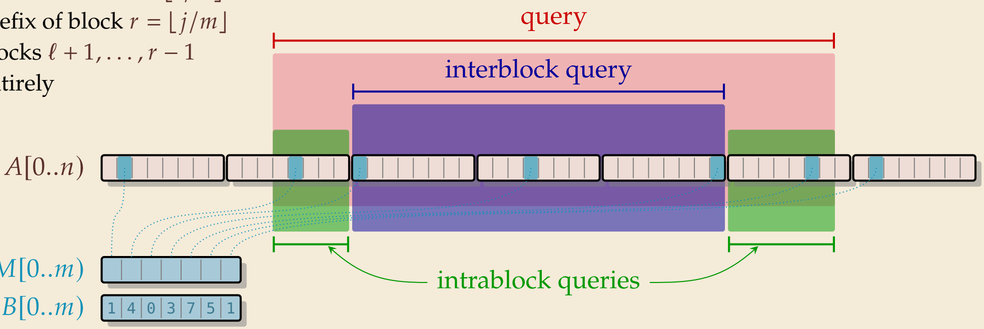
Query decomposition

- 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

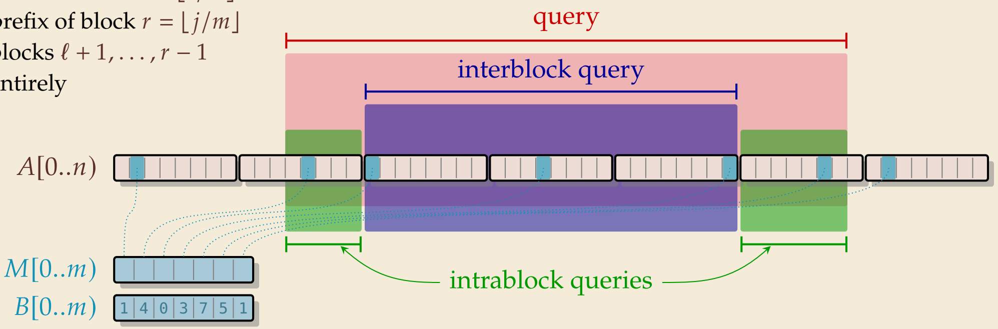
- Query $\mathrm{RMQ}_A(i,j)$ covers
  - suffix of block $\ell = \lfloor i/m \rfloor$
  - prefix of block $r = \lfloor j/m \rfloor$
  - blocks $\ell+1,\ldots,r-1$ entirely



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

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

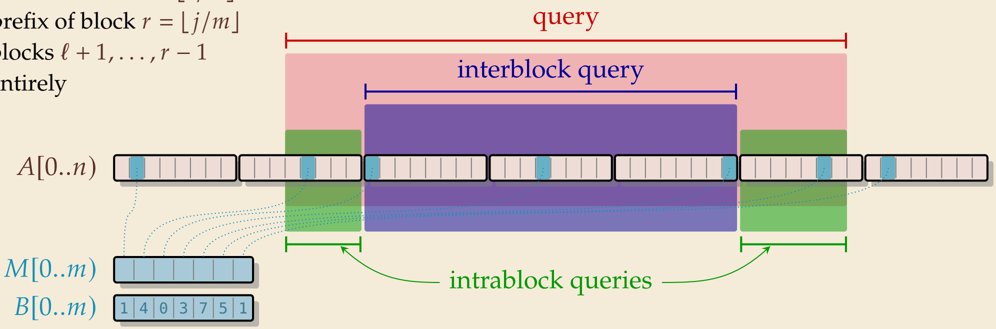


- $\mathrm{RMQ}_A(i,j) \;=\; \operatorname*{arg\,min}_{k\in K} A[k] \quad \text{with} \;\; K \;=\; \left\{ \begin{array}{l} \mathrm{RMQ}_{\text{block } \ell}\big(i-\ell b,\ (\ell+1)b-1\big), \\ b\cdot \mathrm{RMQ}_M\big(\ell+1, r-1\big) + \\ \qquad B\big[\mathrm{RMQ}_M\big(\ell+1, r-1\big)\big], \\ \mathrm{RMQ}_{\text{block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

$\rightsquigarrow$ 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



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

⇝ only 3 possible values to check
if intrablock and interblock queries known

$O(\log n)$

$\langle O(n), O(\log n) \rangle$