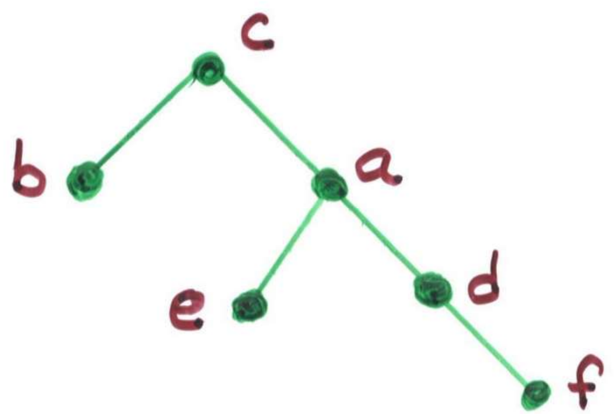
The LCA Problem Revisited

Michael A. Bender
SUNY at Stony Brook

Martín Farach-Colton
Rutgers

Least Common Ancestor (LCA)

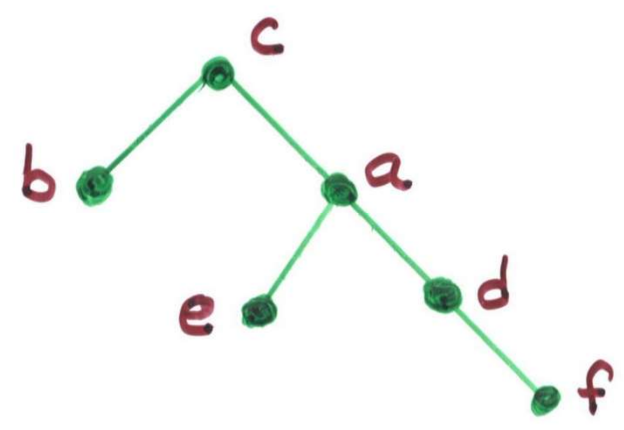
The Least Common Ancestor (LCA) of nodes u and v in a tree is the node farthest from the root that is the ancestor of both u and v.



Example: $LCA(e, f) = a$

Least Common Ancestor (LCA)

The Least Common Ancestor (LCA) of nodes u and v in a tree is the node farthest from the root that is the ancestor of both u and v.



Example: $LCA(e, f) = a$

Problem History

- Famous problem. LCA is the workhorse of many applications.

- Harel and Tarjan, 84. First optimal solution.
  - very complicated and unimplementable.

- Shieber & Vishkin, 88. Simplified LCA.
  - but not simple or particularly implementable.

Problem History

- Famous problem. LCA is the workhorse of many applications.
- Harel and Tarjan, 84. First optimal solution.
  - very complicated and unimplementable.
- Shieber & Vishkin, 88. Simplified LCA.
  - but not simple or particularly implementable.
- Folk wisdom: The LCA is intrinsically complicated.

  (Papers have been written with the sole purpose of avoiding the LCA.)

This Talk

- A truly simple LCA algorithm
  - despite popular belief, the LCA is straightforward and should be used rather than avoided.

This Talk

- A truly simple LCA algorithm
  - despite popular belief, the LCA is straightforward and should be used rather than avoided.

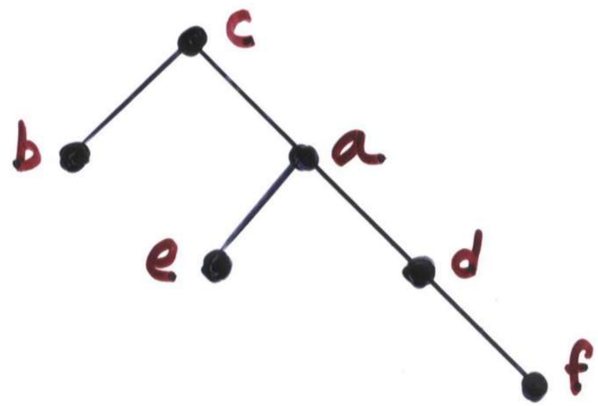
- Unexpected origins: based on a complicated PRAM algorithm [Berkman, Breslauer, Galil, Schieber, Vishkin 89].

Remove PRAM complications $\Longrightarrow$ algorithm is sleek and sequential.

Naive Solution: $\langle O(n^2), O(1) \rangle$

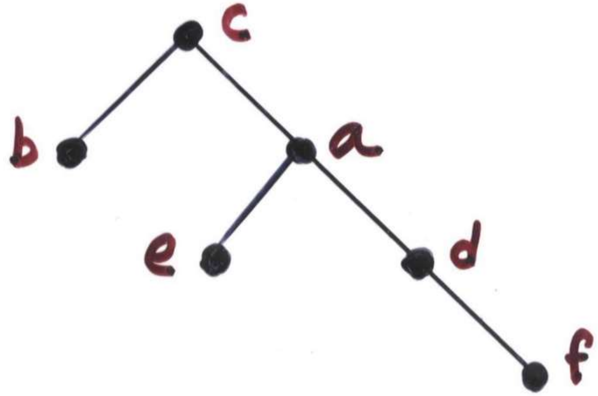
Idea: There are only n^2 possible queries.

Precompute answers to all queries.



Naive Solution: $\langle O(n^2), O(1) \rangle$

Idea: There are only n^2 possible queries.
Precompute answers to all queries.



LCA	a	b	c	d	e	f
a	a	c	c	a	a	a
b	c	b	c	c	c	c
c	c	c	c	c	c	c
d	a	c	c	d	a	d
e	a	c	c	a	e	a
f	a	c	c	d	a	f

Table filled in $O(n^2)$ using dynamic programming.

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

Range Minimum Queries (RMQ)

Given an array $A[1 \dots n]$, $RMQ[i,j]$ returns the index of the smallest element between i and j.

1	2	3	4	5	6	7
17	13	15	10	16	11	12

$RMQ[2,5] = 4$ because $A[4] = 10$ is min value in range.

Range Minimum Queries (RMQ)

Given an array $A[1 \ldots n]$, $RMQ[i,j]$ returns the index of the smallest element between i and j.

1	2	3	4	5	6	7
17	13	15	10	16	11	12

$RMQ[2,5] = 4$ because $A[4] = 10$ is min value in range.

The problem: preprocess $A[1 \ldots n]$ to answer RMQ questions quickly.

- complexity measure: $\langle$preprocess time, query time$\rangle$

Naive Solution for RMQ: $\langle O(n^2), O(1) \rangle$

- There are $O(n^2)$ possible queries.

  Precompute all answers in $O(n^2)$ using dynamic programming.

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

- Same complexities for naïve LCA and naïve RMQ.

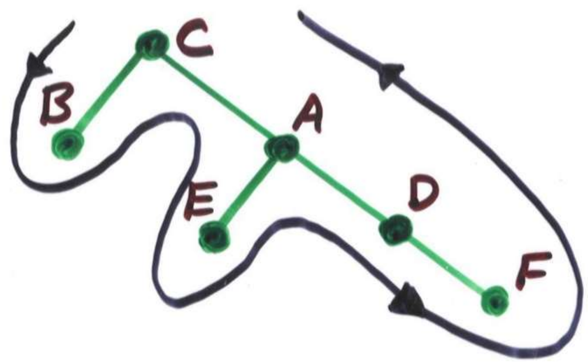
  Is this a coincidence?

No.

It isn't a coincidence.

Reduction from LCA to RMQ

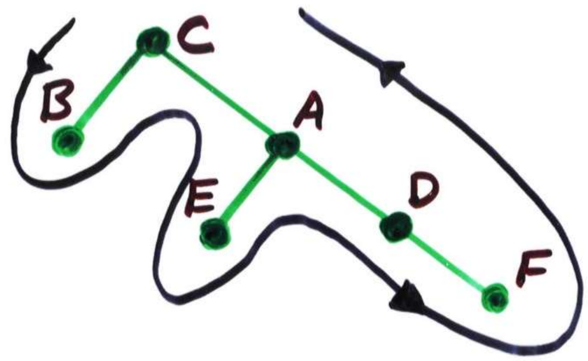
Use Euler Tour/DFS to convert LCA to RMQ.



	1	2	3	4	5	6	7	8	9	10	11
Euler tour E	C	B	C	A	E	A	D	F	D	A	C
Depth of nodes D	0	1	0	1	2	1	2	3	2	1	0

Reduction from LCA to RMQ

Use Euler Tour/DFS to convert LCA to RMQ.



	1	2	3	4	5	6	7	8	9	10	11
Euler tour E	C	B	C	A	E	A	D	F	D	A	C
Depth of nodes D	0	1	0	1	2	1	2	3	2	1	0

$\boxed{\text{RMQ}}$

LCA[E,F] = A

Find first locations of E and F in Euler tour.

RMQ between these locations in Depth Array ⇒ A.

Rest of Talk

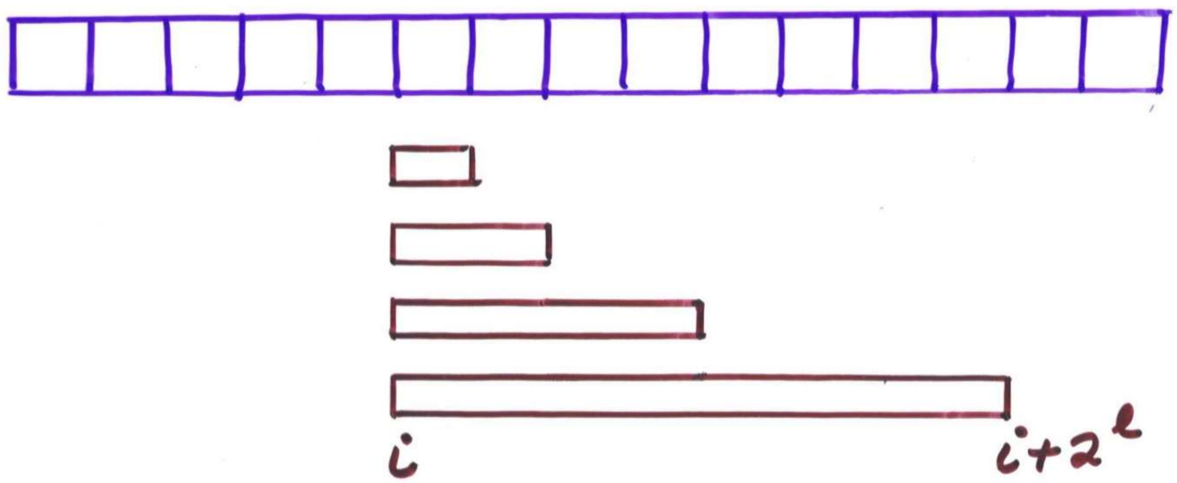
From now on we focus on the RMQ Problem.
We use a solution to RMQ to solve LCA.

Approach: improve the naïve $\langle O(n^2), O(1) \rangle$ solution in stages.

$O(n \log n)$ Preprocessing

Idea: only store RMQ for ranges whose sizes are powers of 2.

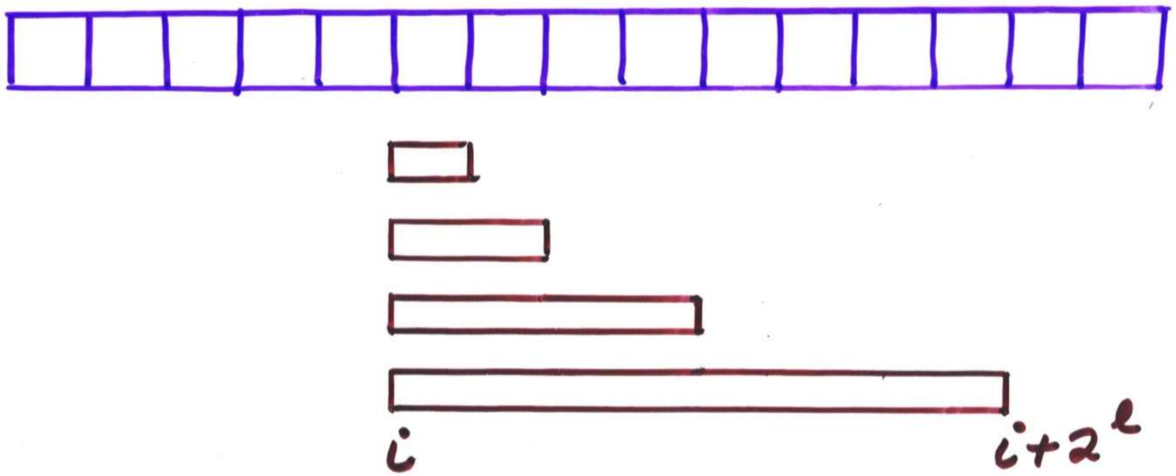
E.g., for $i = 1 \ldots n$ and $\ell = 0 \ldots \lfloor \log n \rfloor$, store $RMQ[i, i+2^\ell]$.



$O(n \log n)$ Preprocessing

Idea: only store RMQ for ranges whose sizes are powers of 2.

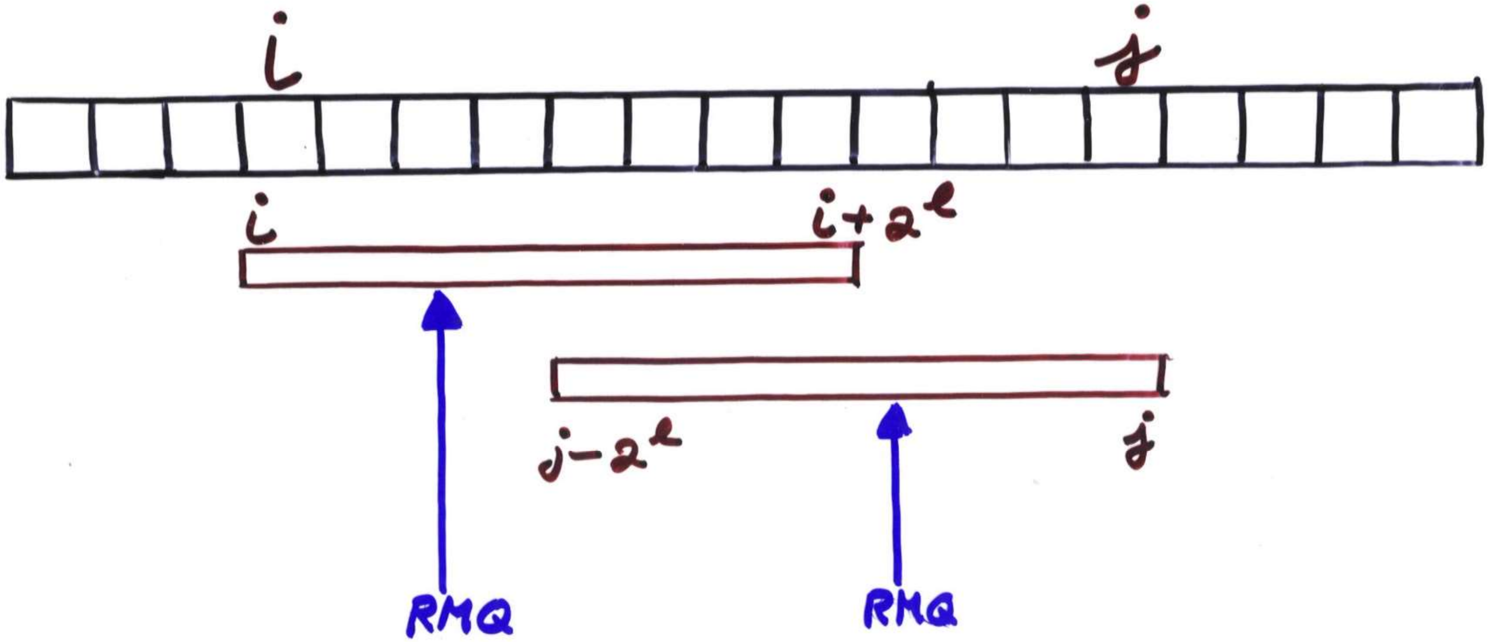
E.g., for $i = 1 \ldots n$ and $\ell = 0 \ldots \lfloor \log n \rfloor$, store RMQ$[i, i+2^\ell]$.



Queries can be answered in $O(1)$!

$O(n \log n)$ Preprocessing

$RMQ[i,j]$ can be found by taking a minimum of 2 values.

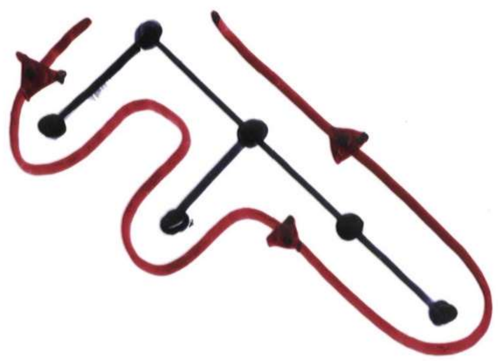


$\Rightarrow O(1)$ queries.

Towards a $\langle O(n), O(1) \rangle$ Algorithm

To improve the LCA, observe that the RMQs that we generate have a special structure:

- ± 1 RMQ. All neighbors differ by ± 1.

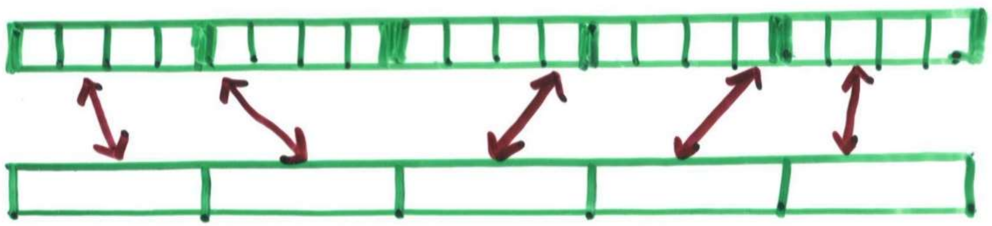


0	1	0	1	2	1	2	3	2	1	0

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

Break array into groups of size $\frac{1}{2}\log n$.

$O(n)$-size array



$O(\frac{n}{\log n})$-array.

store max of each group.

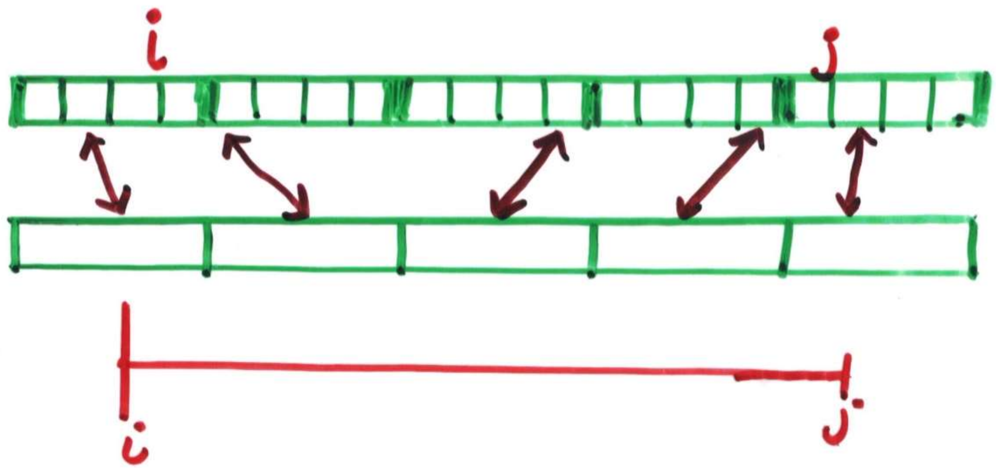
Towards $\langle O(n), O(1) \rangle$

Break array into groups of size $\frac{1}{2}\log n$.

$O(n)$-size array

$O\left(\frac{n}{\log n}\right)$-array.

store max of each group.



The RMQ either resides in a completely covered group or in a partially covered group.

$\Rightarrow$ Compute RMQ in $\frac{2n}{\log n}$ array and in each $\frac{\log n}{2}$ array. Take min of all possibilities.

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

- preprocessing for $O(\frac{n}{\log n})$ array:

$$O\left(\frac{n}{\log n} \cdot \log\left(\frac{n}{\log n}\right)\right) = O(n)$$

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

- preprocessing for $O(\frac{n}{\log n})$ array:

$$O\left(\frac{n}{\log n} \cdot \log\left(\frac{n}{\log n}\right)\right) = O(n)$$

- preprocessing for $O(\frac{n}{\log n})$ groups of size $O(\log n)$:

$$O\left(\frac{n}{\log n}\right) \cdot O(\log n \cdot \log\log n) = O(n \log\log n)$$

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

- preprocessing for $O(\frac{n}{\log n})$ array:

$$O\left(\frac{n}{\log n} \cdot \log\left(\frac{n}{\log n}\right)\right) = O(n)$$

- preprocessing for $O(\frac{n}{\log n})$ groups of size $O(\log n)$:

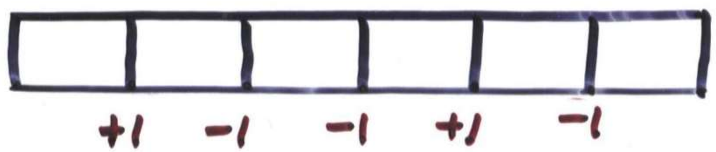
$$O\left(\frac{n}{\log n}\right) \cdot O(\log n \cdot \log\log n) = O(n \log\log n)$$

$\Longrightarrow$ Closer to $O(n)$ but not there yet!

Improving ±1 RMQ in Small Arrays

Use ±1 structure! RMQ problem completed determined by pattern of +1's and −1's.

$\frac{\log n}{2}$ array:



Improving ±1 RMQ in Small Arrays

Use ±1 structure! RMQ problem completed determined by pattern of +1's and −1's.

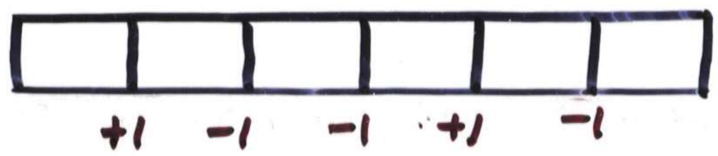
$\frac{\log n}{2}$ array:

$\Rightarrow$ only $2^{\frac{1}{2}\log n - 1} = \frac{1}{2}\sqrt{n}$ distinct RMQ problems.

Improving ±1 RMQ in Small Arrays

Use ±1 structure! RMQ problem completed determined by pattern of +1's and −1's.

$\frac{\log n}{2}$ array:

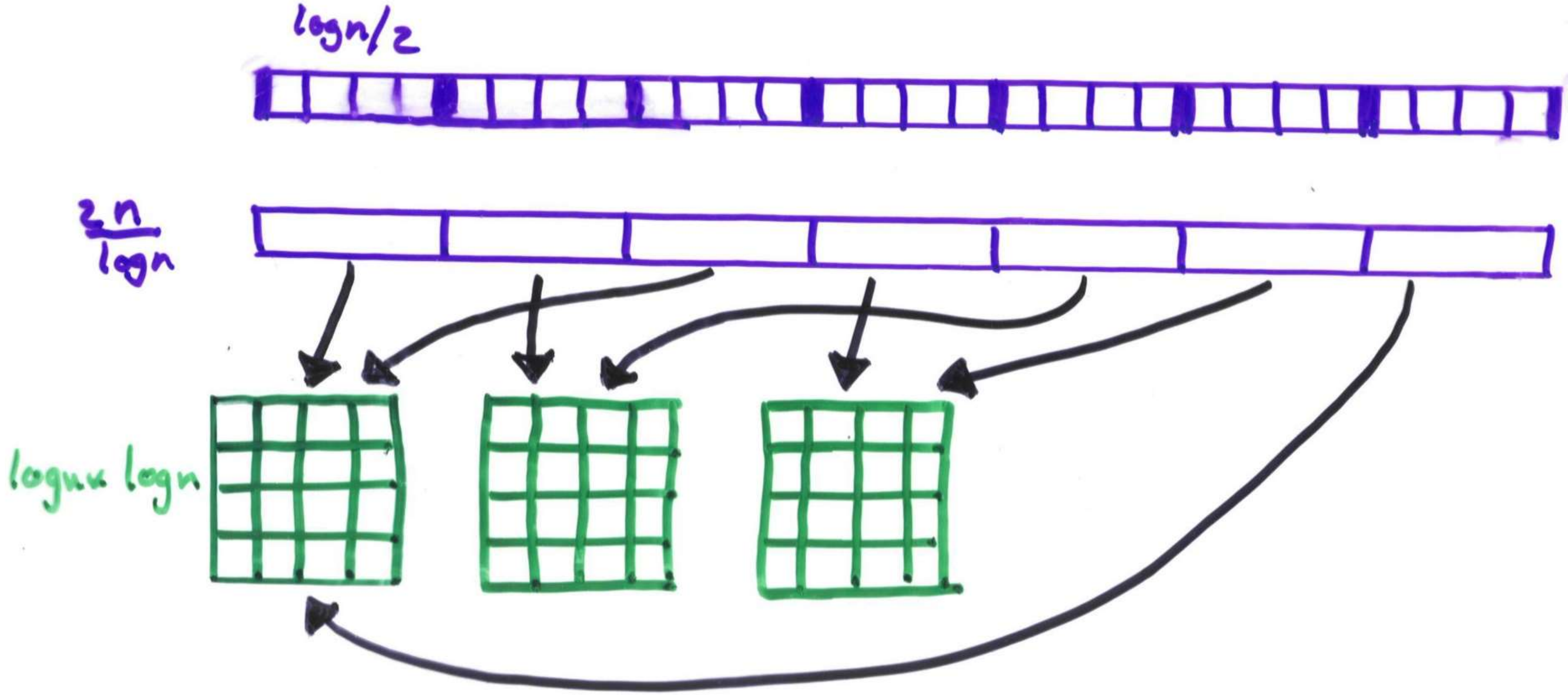


$\Rightarrow$ only $2^{\frac{1}{2}\log n - 1} = \frac{1}{2}\sqrt{n}$ distinct RMQ problems.

Precompute all possible small RMQ problems in

$$O(\sqrt{n})\cdot O(\log^2 n) = O(\sqrt{n}\log^2 n).$$

$\langle O(n), O(1) \rangle$ LCA / ± 1 RMQ

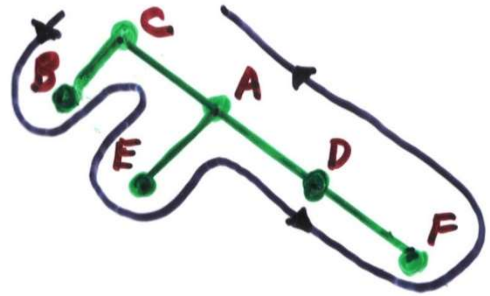


$O(n) + O(\sqrt{n} \log^2 n) = O(n)$ preprocessing

Queries answered by taking min of 4 numbers.

Reduction from LCA to RMQ

Use Euler Tour/DFS to convert LCA to RMQ.



Euler Tour E

1	2	3	4	5	6	7	8	9	10	11
C	B	C	A	E	A	D	F	D	A	C

Depth of Nodes D

0	1	0	1	2	1	2	3	2	1	0

Representative R
(first time node appears in DFS)

A	B	C	D	E	F
4	2	1	7	5	8

$$LCA(x,y) = E[RMQ_{\text{Depth } D}(R[x], R[y])]$$

Arbitrary RMQ

RMQ

$\Downarrow$ $O(n)$ reduction using Cartesian Trees.

LCA

$\Downarrow$ $O(n)$ reduction using Euler Tour.

± 1 RMQ

$\Downarrow$

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