

Range Minimum Queries and Lowest Common Ancestor

Inge Li Gørtz

Range Minimum Queries and Lowest Common Ancestor

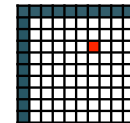
- Range Minimum Queries (RMQ) and Lowest Common Ancestor (LCA)
- RMQ
 - Simple solutions
 - Better solution
 - 2-level solution
- Reduction between RMQ and LCA
- Dynamic RMQ

Range Minimum Queries

- **Range minimum query problem.** Preprocess array $A[1 \dots n]$ of integers to support
 - $\text{RMQ}(i,j)$: return the (entry of) minimum element in $A[i \dots j]$.

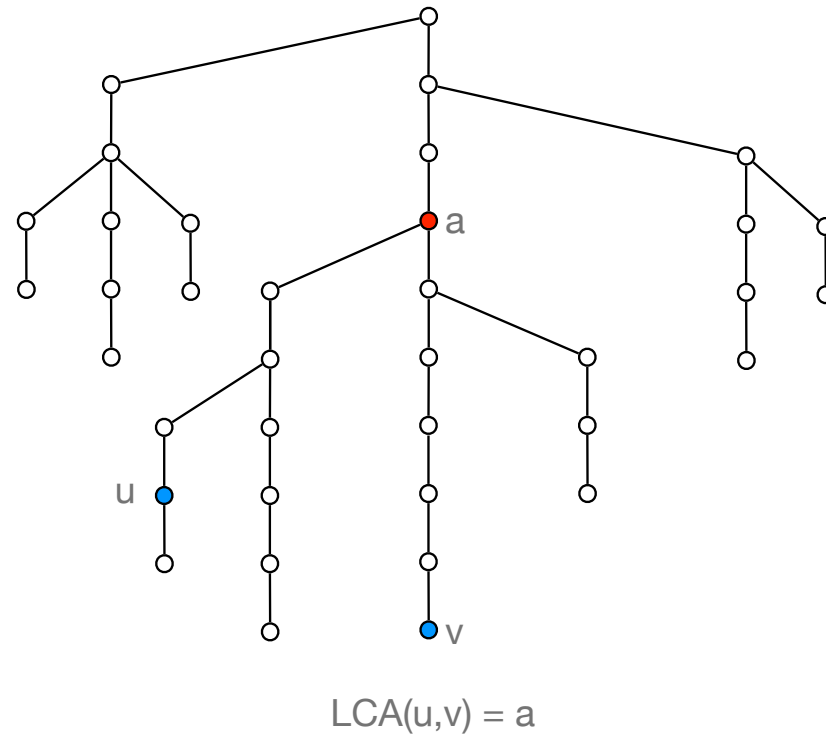
0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3

- $\text{RMQ}(2,5) = 2$ (index 4)
- Basic (extreme) solutions
 - **Linear search:**
 - Space: $O(n)$. Only keep array (no extra space)
 - Time: $O(j-i) = O(n)$
 - **Save all possible answers:** Precompute and save all answers in a table.
 - Space: $O(n^2)$ pairs $\Rightarrow O(n^2)$ space
 - Time: $O(1)$



Lowest Common Ancestor

- **Lowest common ancestor problem.** Preprocess rooted tree T with n nodes to support
 - $LCA(u,v)$: return the lowest common ancestor of u and v .



Lowest Common Ancestor

- Basic (extreme) solutions
 - **Linear search**: Follow paths to root and mark when you visit a node.
 - Space: $O(n)$. Only keep tree (no extra space)
 - Time: $O(\text{depth of tree}) = O(n)$
 - **Save all possible answers**: Precompute and save all answers in a table.
 - Space: $O(n^2)$ pairs $\Rightarrow O(n^2)$ space
 - Time: $O(1)$

RMQ and LCA

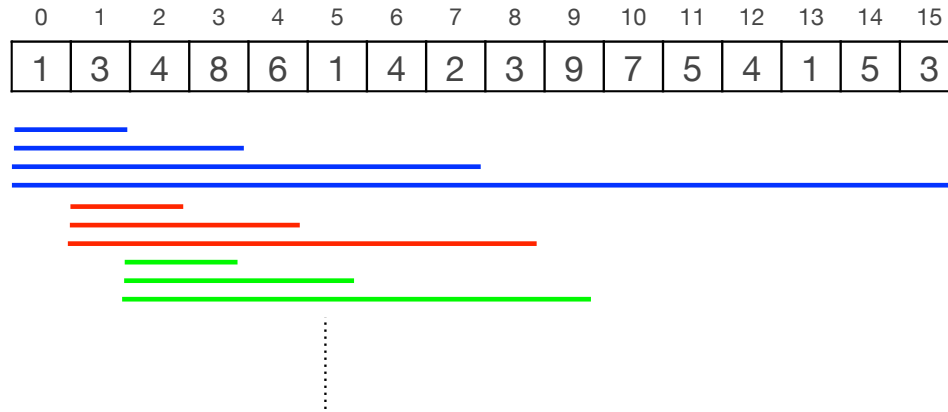
- **Outline.**
 - Can solve both RMQ and LCA in linear space and constant time.
 - First solution to RMQ
 - Solution to a special case of RMQ.
 - See that RMQ and LCA are equivalent (can reduce one to the other both ways).

RMQ

Sparse table solution

RMQ: Sparse table solution

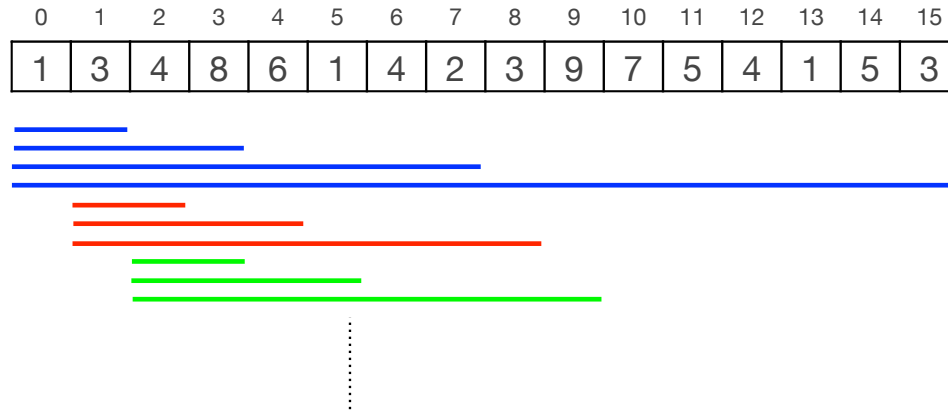
- Save the result for all intervals of length a power of 2.



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

RMQ: Sparse table solution

- Save the result for all intervals of length a power of 2.



- Space: $O(n \log n)$

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

RMQ: Sparse table solution

- Save the result for all intervals of length a power of 2.

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



RMQ(6, 12) = ?

- Space: $O(n \log n)$

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

RMQ: Sparse table solution

- Save the result for all intervals of length a power of 2.

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



$$\text{RMQ}(6, 12) = \min(\text{RMQ}(6,9), \text{RMQ}(9,12)) = \min(2,4) = 2$$

- Space: $O(n \log n)$

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

RMQ: Sparse table solution

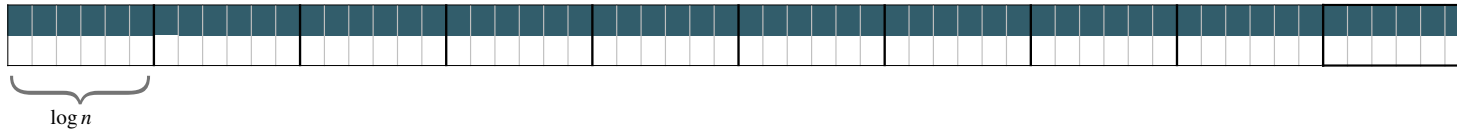
- Query:



- Any interval is the union of two power of 2 intervals.
 - k largest number such that $2^k \leq j - i + 1$.
 - Lookup results for the two intervals and take minimum.
- Time: $O(1)$
- Space: $O(n \log n)$
- Preprocessing time: $O(n \log n)$
 - To compute results for length 2^i use results for length 2^{i-1} .

Reducing Space

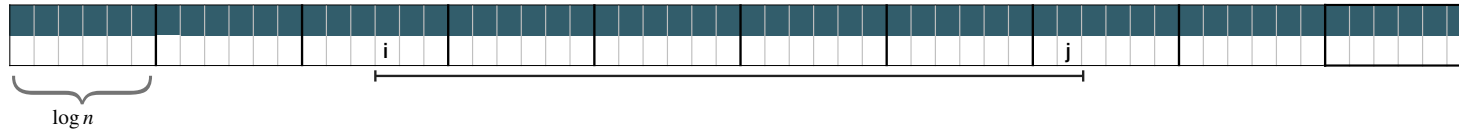
- Divide A into blocks of size $\log n$



- 2-level data structure:
 - A data structure on minimums of the blocks
 - A data structure for queries inside blocks.

Reducing Space

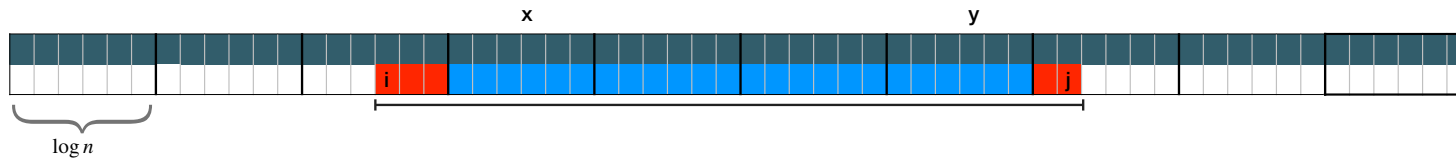
- Divide A into blocks of size $\log n$



- 2-level data structure:
 - A data structure on minimums of the blocks
 - A data structure for queries inside blocks.

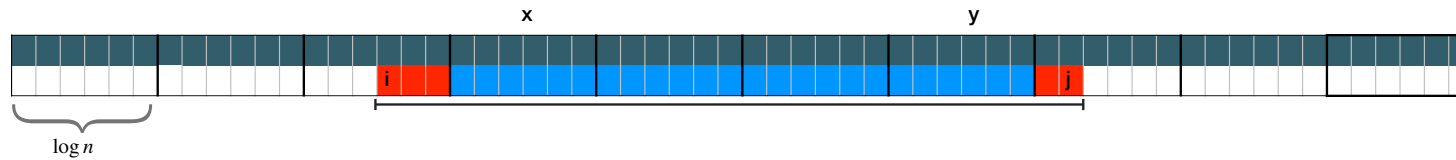
Reducing Space

- Divide A into blocks of size $\log n$



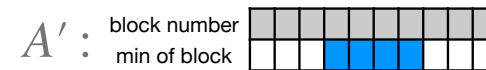
- 2-level data structure:
 - A data structure on minimums of the blocks
 - A data structure for queries inside blocks.
- $\text{RMQ}(i,j) = \min\{ \text{RMQ on blocks } x \text{ to } y, \text{RMQ inside block } x-1, \text{RMQ inside block } y+1 \}$.

Reducing Space: Data Structure on Blocks

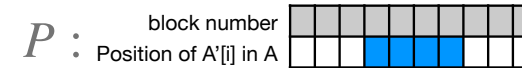


- Two new arrays.

- Array A' : minimum from each block



- P : position in A where $A'[i]$ occurs.



- Sparse table data structure on A' .

- Space: $O(|A'| \log |A'|) = O\left(\frac{n}{\log n} \cdot \log \frac{n}{\log n}\right) = O(n)$.

- Time: $O(1)$

Inside blocks

- Linear search inside blocks:
 - $O(\log n)$ query time.

$\pm 1\text{RMQ}$

RMQ: Linear space

- Consider ± 1 RMQ: consecutive entries differ by 1.

0	1	2	3	4	5	6	7	8	9	10	11	12
4	5	6	5	4	3	2	3	2	3	4	5	4

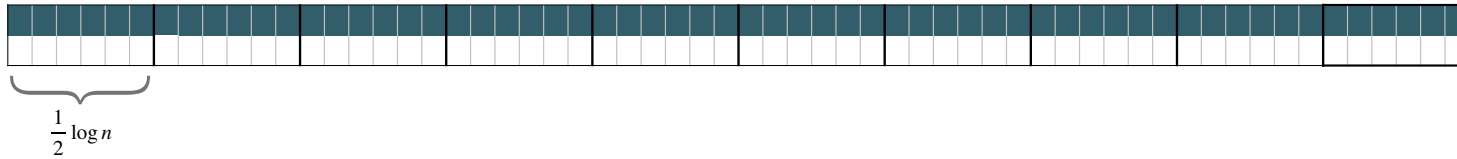
- 2-level solution: Combine
 - $O(n \log n)$ space, $O(1)$ time
 - $O(n^2)$ space, $O(1)$ time.

⇓

 - $O(n)$ space, $O(1)$ time.

± 1 RMQ

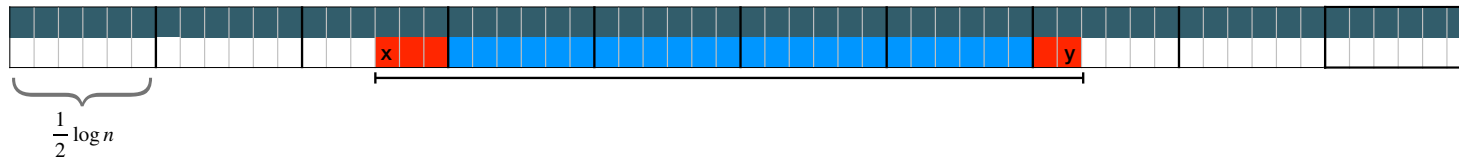
- Divide A into blocks of size $\frac{1}{2} \log n$



- 2-level data structure:
 - Sparse table on blocks
 - Tabulation inside blocks.

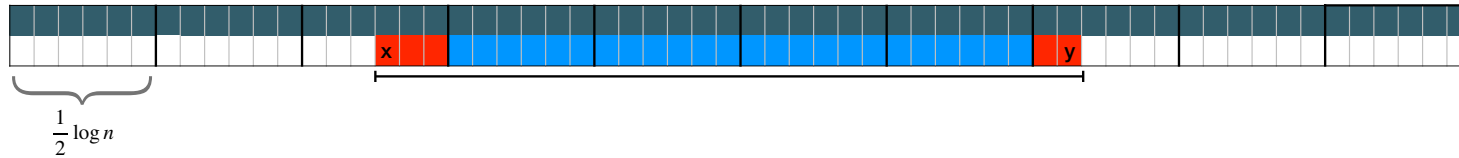
± 1 RMQ

- Divide A into blocks of size $\frac{1}{2} \log n$

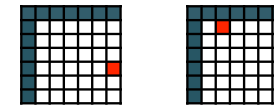


- 2-level data structure:
 - Sparse table on blocks
 - Tabulation inside blocks.
- $\text{RMQ}(x,y) = \min\{ \text{RMQ on blocks } i \text{ to } j, \text{RMQ inside block } i-1, \text{RMQ inside block } j+1 \}$.

± 1 RMQ: Data structure inside blocks



- Precompute and save all answers for each block.
- Gives solution using
 - Space: $O(n)$ + space for precomputed tables.
 - Time: $O(1) + O(1) + O(1) = O(1)$.



2 table
lookups

sparse
table

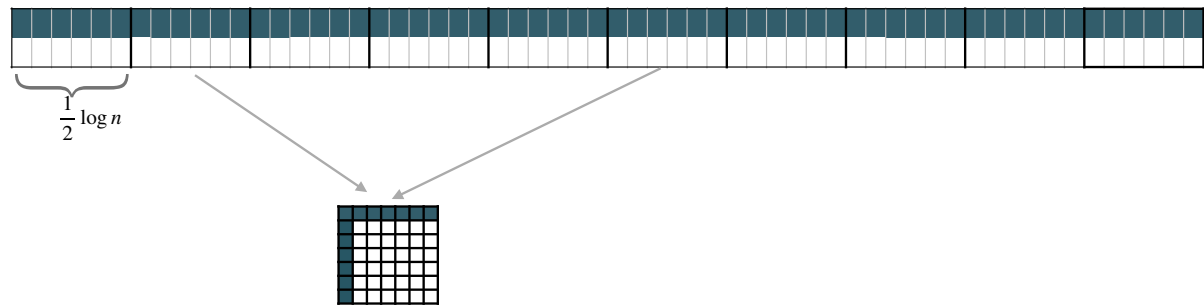
$\min\{.,.,.\}$

± 1 RMQ: Storing the precomputed tables

- Naively: $\log^2 n$ for each table $\Rightarrow n \log n$ space. 😞
- **Observation:** If $X[i] = Y[i] + c$ then all RMQ answers are the same for X and Y .
 - $X = [7, 6, 5, 6, 5, 4]$
 - $Y = [3, 2, 1, 2, 1, 0]$
- Describe block by sequence of $+1$ s and -1 s:
 - $X - Y = [-1, -1, +1, -1, -1]$.
- How many different block descriptions are there?
 - length of sequence = $\frac{1}{2} \log n - 1$
 - #sequences = $2^{\frac{1}{2} \log n - 1} \leq \sqrt{n}$.

± 1 RMQ: Data structure inside blocks

- Precompute and save all answers for each normalized block.
- Size of a table: $O(\log^2 n)$
- For each block save which precomputed table it uses.

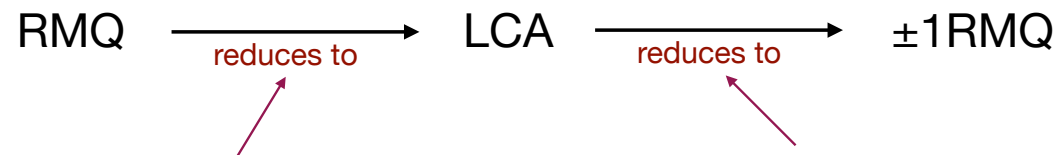


- Space: $O(\sqrt{n} \cdot \log^2 n) + O(n/\log n) = O(n)$
- Plugging into 2-level solution:
 - Space: $O(n)$ + space for precomputed tables = $O(n)$.

LCA and RMQ

RMQ and LCA

- We will show

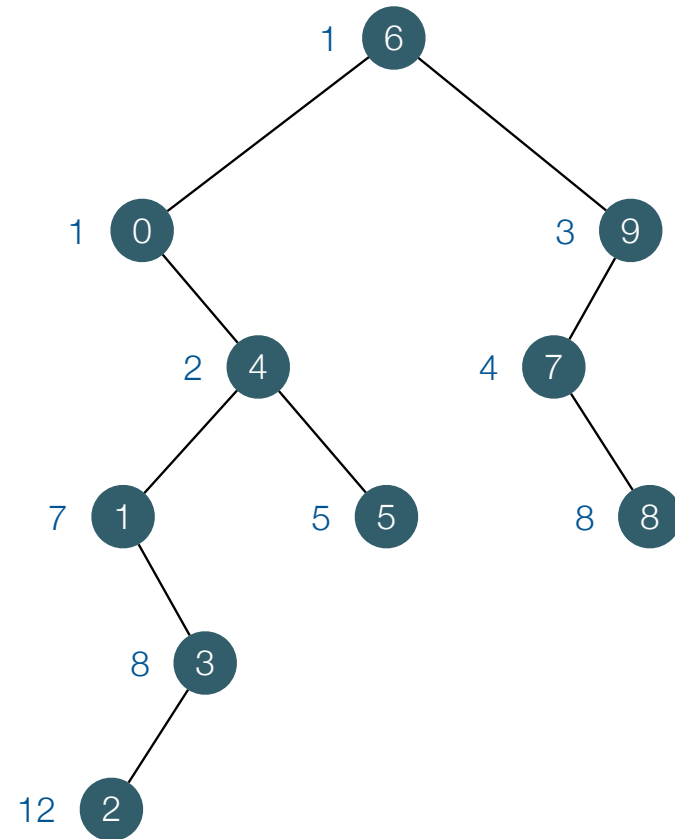


If there is a solution to LCA using $s(n)$ space and $t(n)$ time, then there is a solution to RMQ using $O(s(n))$ space and $O(t(n))$ time.

If there is a solution to ± 1 RMQ using $s(n)$ space and $t(n)$ time, then there is a solution to LCA using $O(s(n))$ space and $O(t(n))$ time.

RMQ to LCA: Cartesian Tree

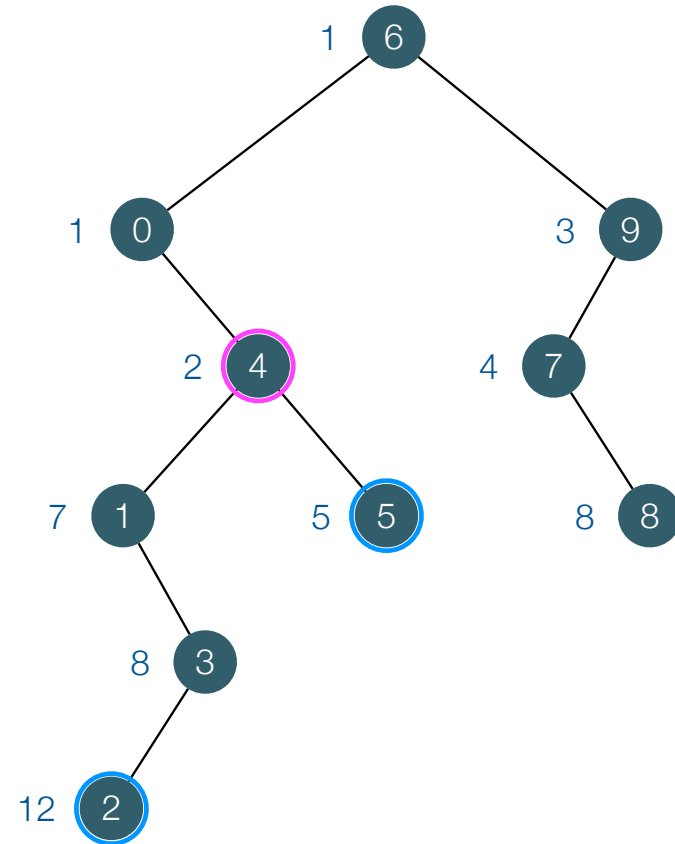
0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3



RMQ to LCA: Cartesian Tree

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3

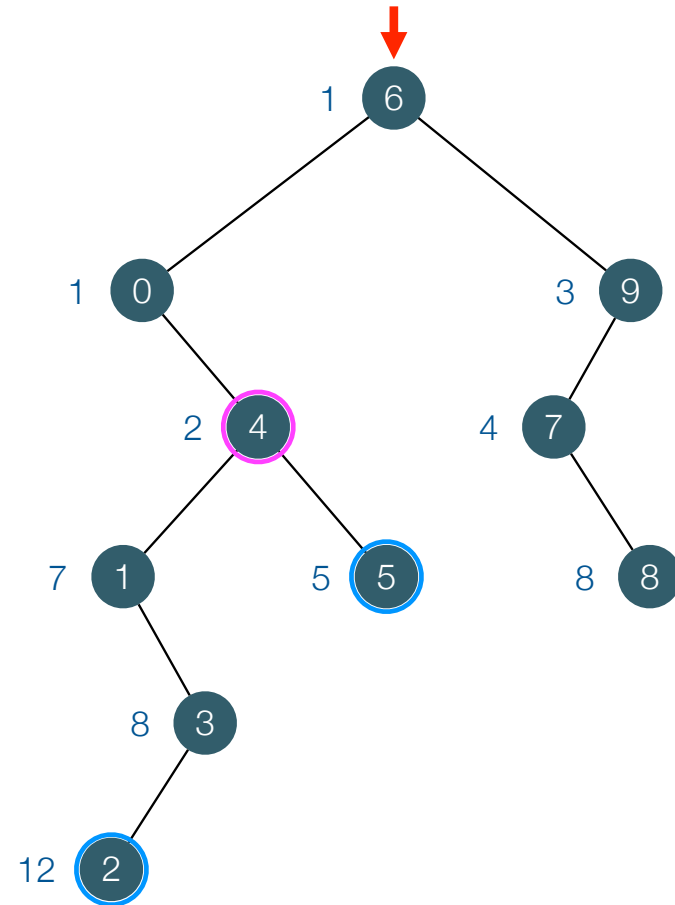
- RMQ(2,5)



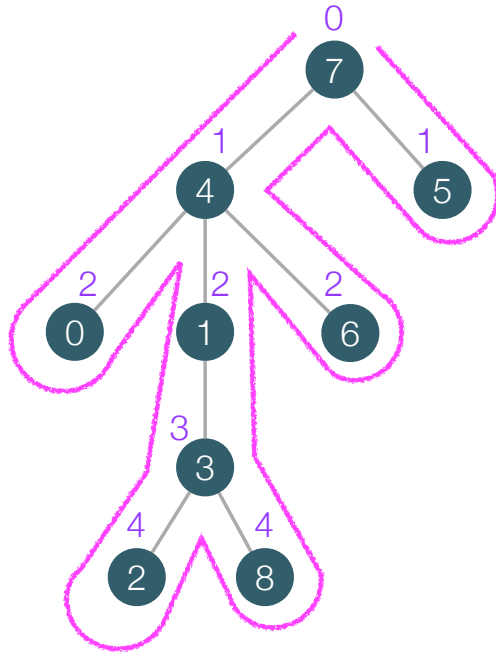
RMQ to LCA: Cartesian Tree

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3

- $\text{RMQ}(2,5) = \text{LCA}(2,5)$



LCA to ± 1 RMQ



- E =

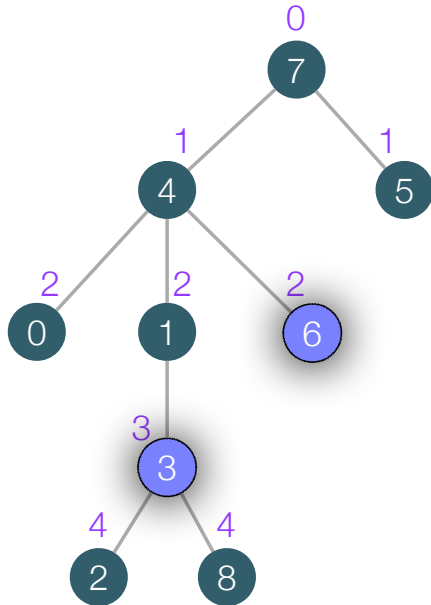
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
7	4	0	4	1	3	2	3	8	3	1	4	6	4	7	5	7
- A =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	2	1	2	3	4	3	4	3	2	1	2	1	0	1	0
- R =

0	1	2	3	4	5	6	7	8
2	4	6	5	1	15	12	0	8

- **E**: Euler tour representation. preorder walk, write id of node when met.
- **A**: depth of node node in E[i].
- **R**: first occurrence in E of node with id i
- **LCA**(i, j) = E[RMQ_A(R[i], R[j])].

LCA to ± 1 RMQ



- E =

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

- A =

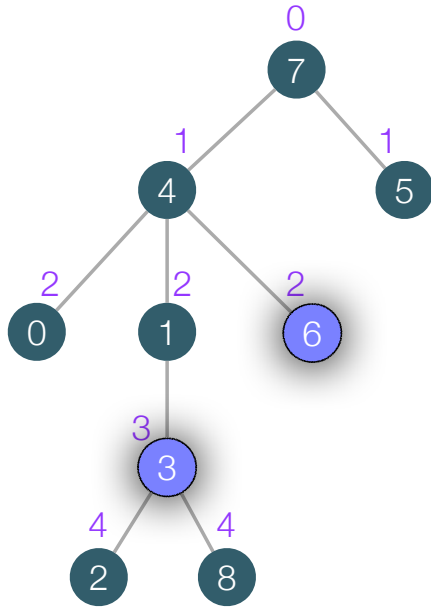
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	2	1	2	3	4	3	4	3	2	1	2	1	0	1	0

- R =

0	1	2	3	4	5	6	7	8
2	4	6	5	1	15	12	0	8

- **E**: Euler tour representation. preorder walk, write id of node when met.
- **A**: depth of node node in E[i].
- **R**: first occurrence in E of node with id i
- **LCA**(i, j) = E[RMQ_A(R[i], R[j])].

LCA to ± 1 RMQ



• E =

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

• A =

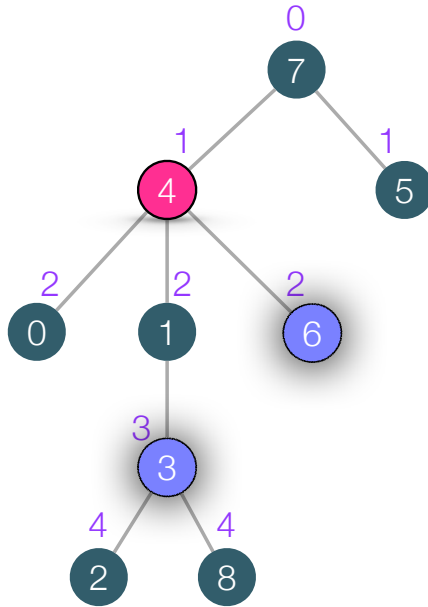
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	2	1	2	3	4	3	4	3	2	1	2	1	0	1	0

• R =

0	1	2	3	4	5	6	7	8
2	4	6	5	1	15	12	0	8

- **E**: Euler tour representation. preorder walk, write id of node when met.
- **A**: depth of node node in $E[i]$.
- **R**: first occurrence in E of node with id i
- **LCA**(i, j) = $E[\text{RMQ}_A(R[i], R[j])]$.

LCA to ± 1 RMQ



• E =

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

$|E| = 2n - 1$

• A =

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

$|A| = 2n - 1$

• R =

0	1	2	3	4	5	6	7	8
2	4	6	5	1	15	12	0	8

$|R| = n$

Space $O(n)$:

- 3 tables
- ± 1 RMQ data structure on table of length $2n$

- **E**: Euler tour representation. preorder walk, write id of node when met.
- **A**: depth of node node in $E[i]$.
- **R**: first occurrence in E of node with id i
- **LCA**(i, j) = $E[\text{RMQ}_A(R[i], R[j])]$.

RMQ and LCA



If there is a solution to LCA using $s(n)$ space and $t(n)$ time, then there is a solution to RMQ using $O(s(n))$ space and $O(t(n))$ time.

If there is a solution to ± 1 RMQ using $s(n)$ space and $t(n)$ time, then there is a solution to LCA using $O(s(n))$ space and $O(t(n))$ time.

- **Theorem.** RMQ and LCA can be solved in $O(n)$ space and $O(1)$ query time.