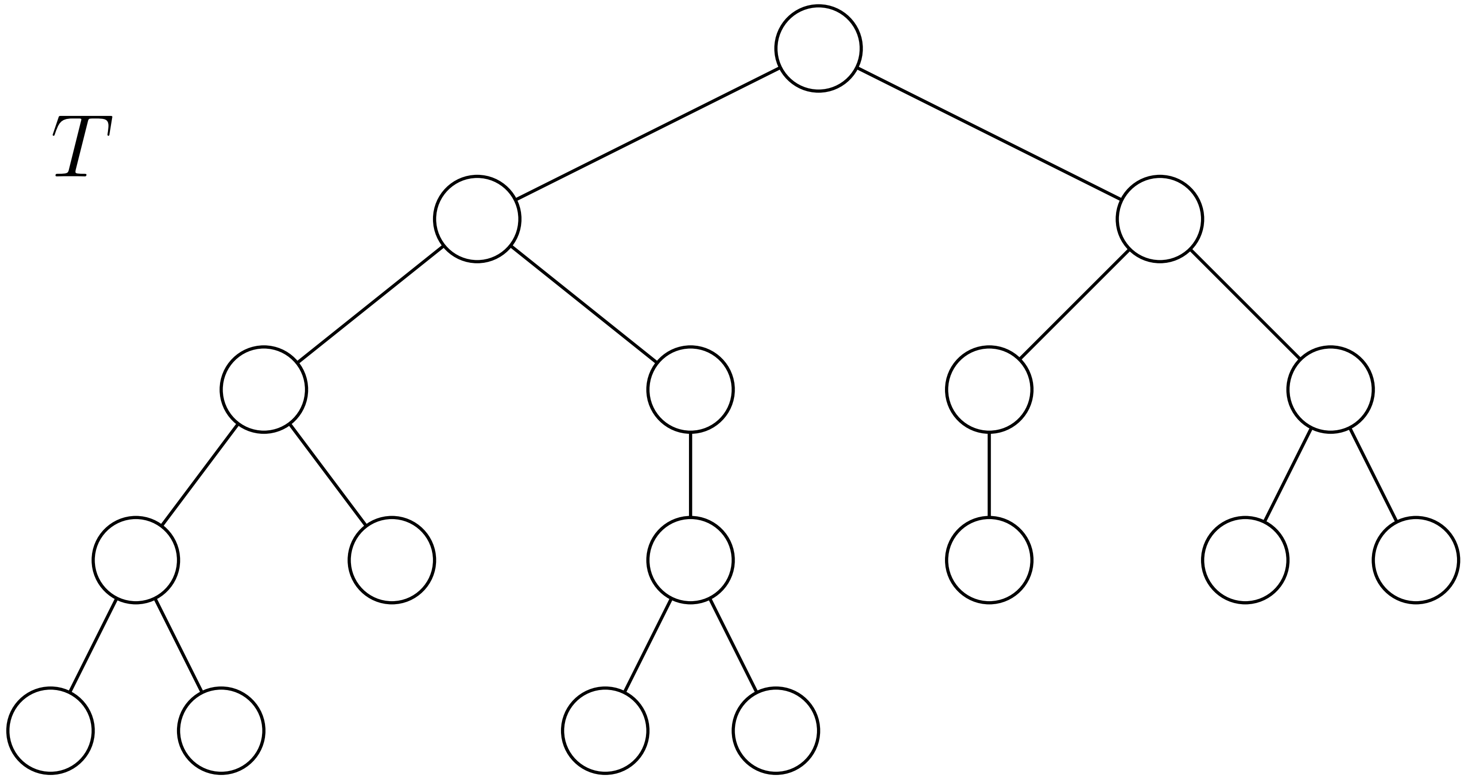
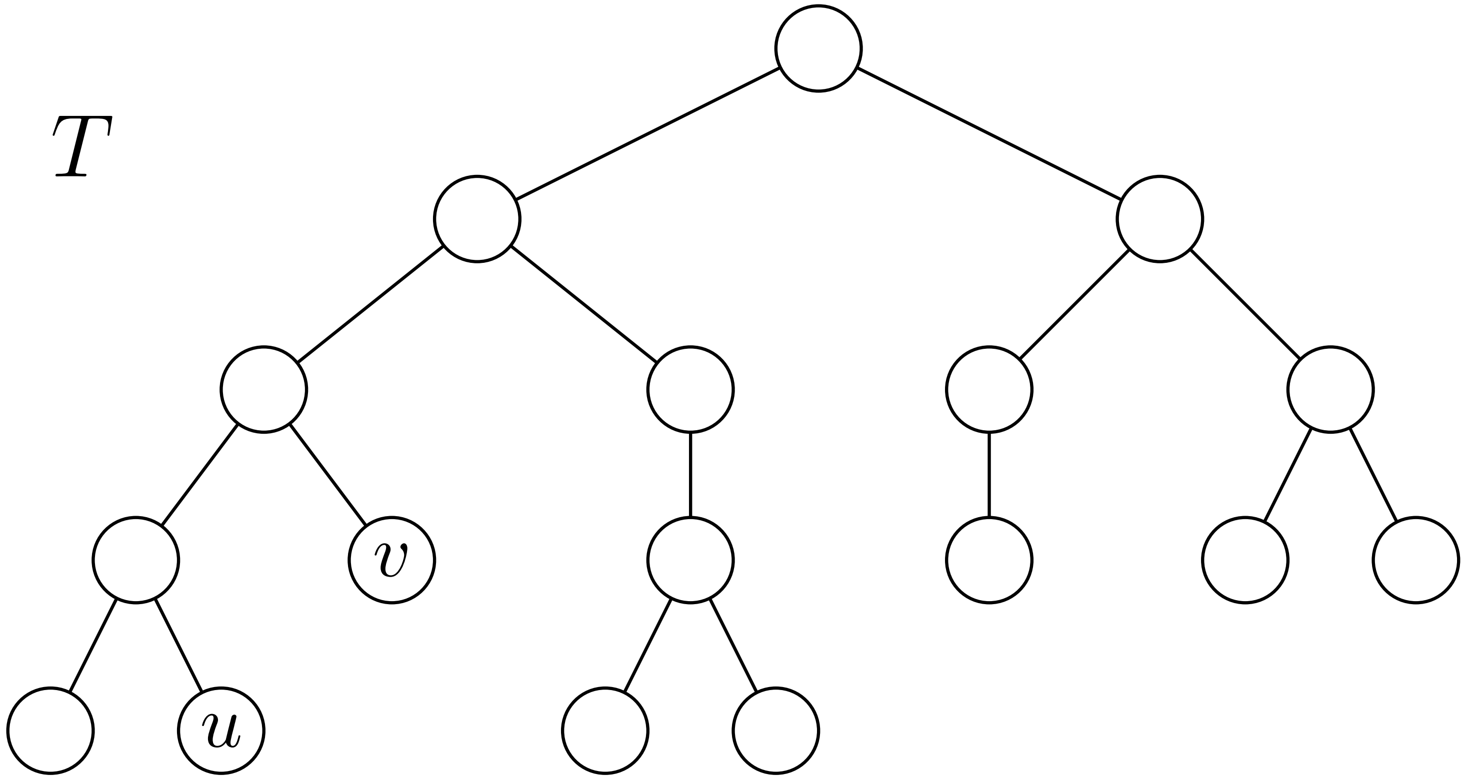
Lowest Common Ancestor Queries

Lowest Common Ancestors



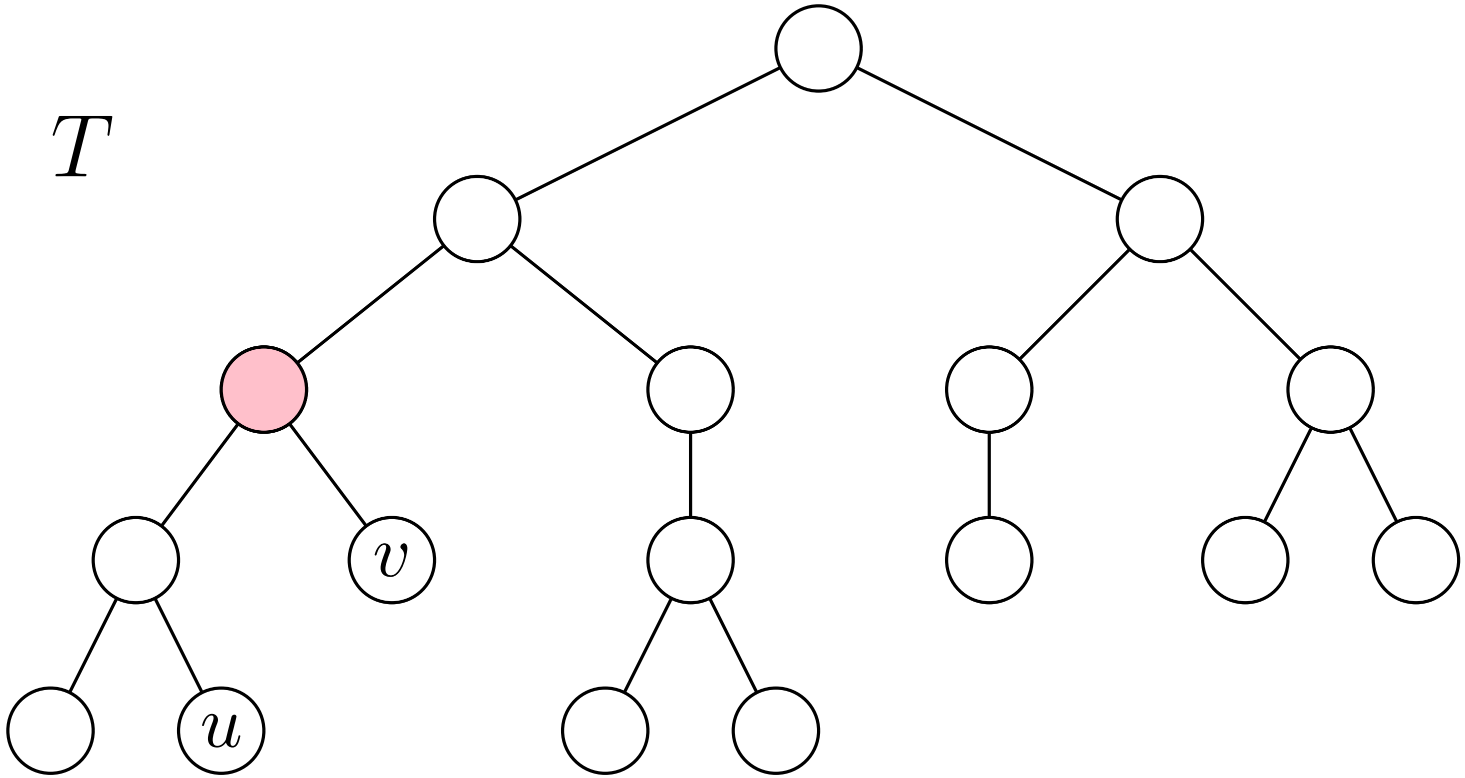
Definition: The *lowest common ancestor* $\mathsf{LCA}_T(u, v)$ of u and v in a rooted tree T is the vertex of maximum depth that is an ancestor of both u and v.

Lowest Common Ancestors



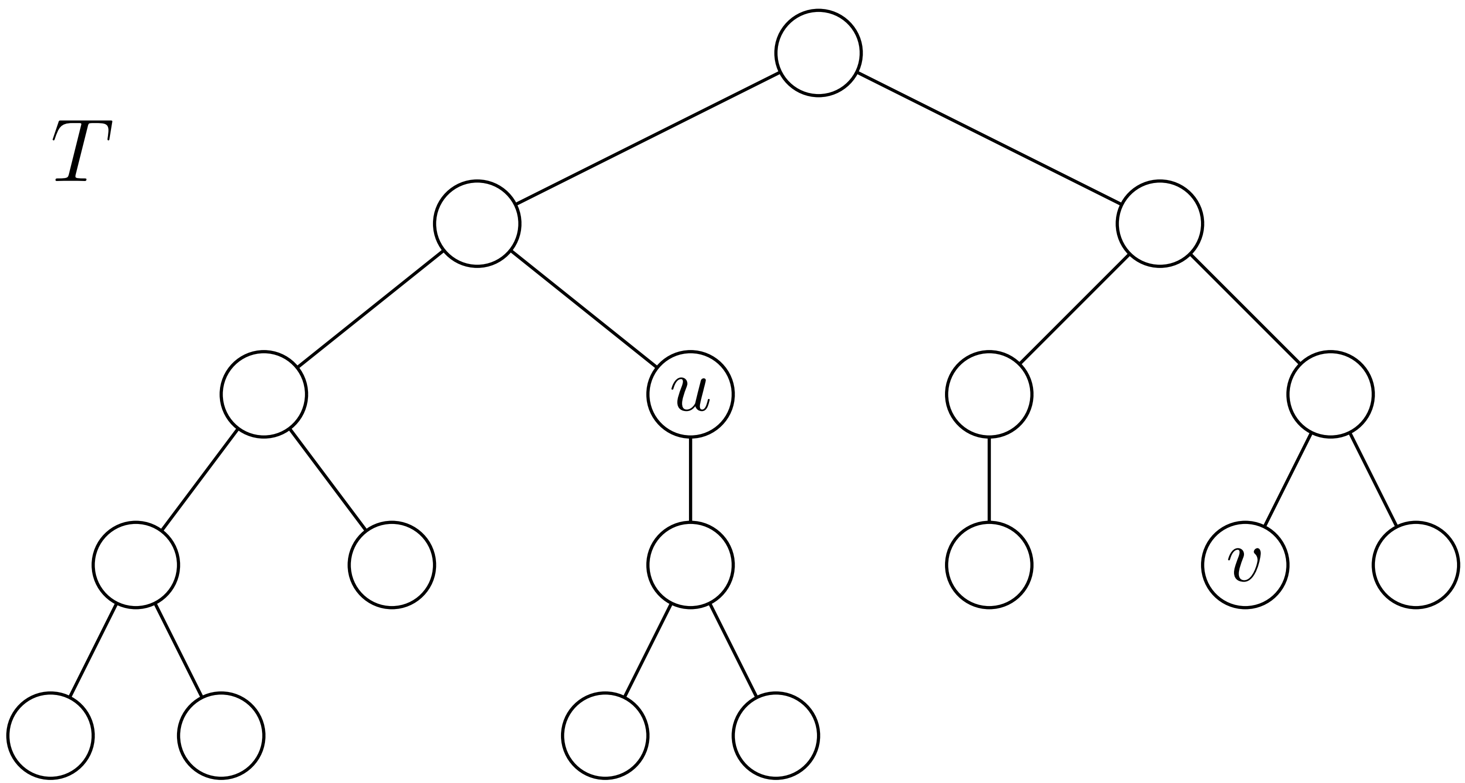
Definition: The *lowest common ancestor* $\mathrm{LCA}_T(u, v)$ of u and v in a rooted tree T is the vertex of maximum depth that is an ancestor of both u and v.

Lowest Common Ancestors



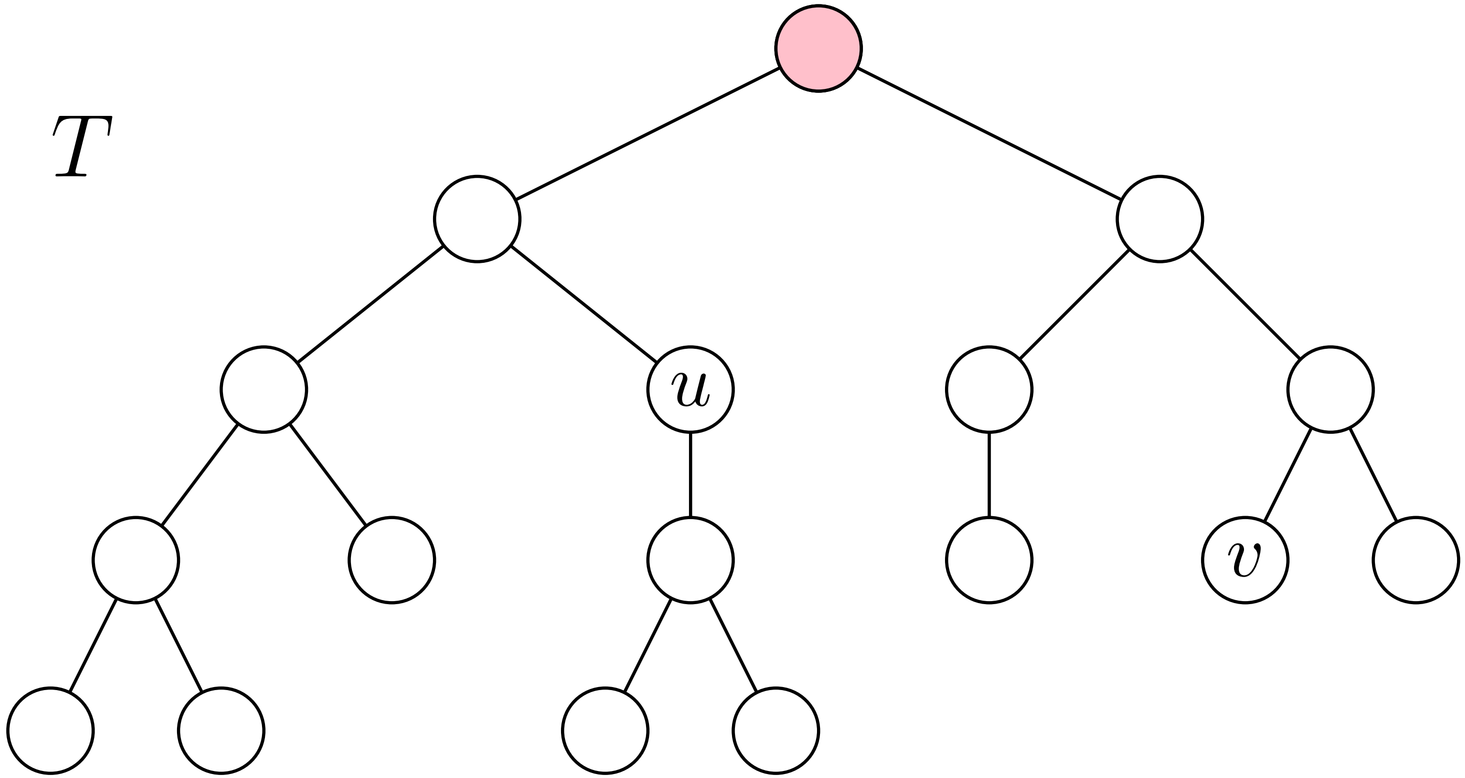
Definition: The *lowest common ancestor* $\text{LCA}_T(u, v)$ of u and v in a rooted tree T is the vertex of maximum depth that is an ancestor of both u and v.

Lowest Common Ancestors



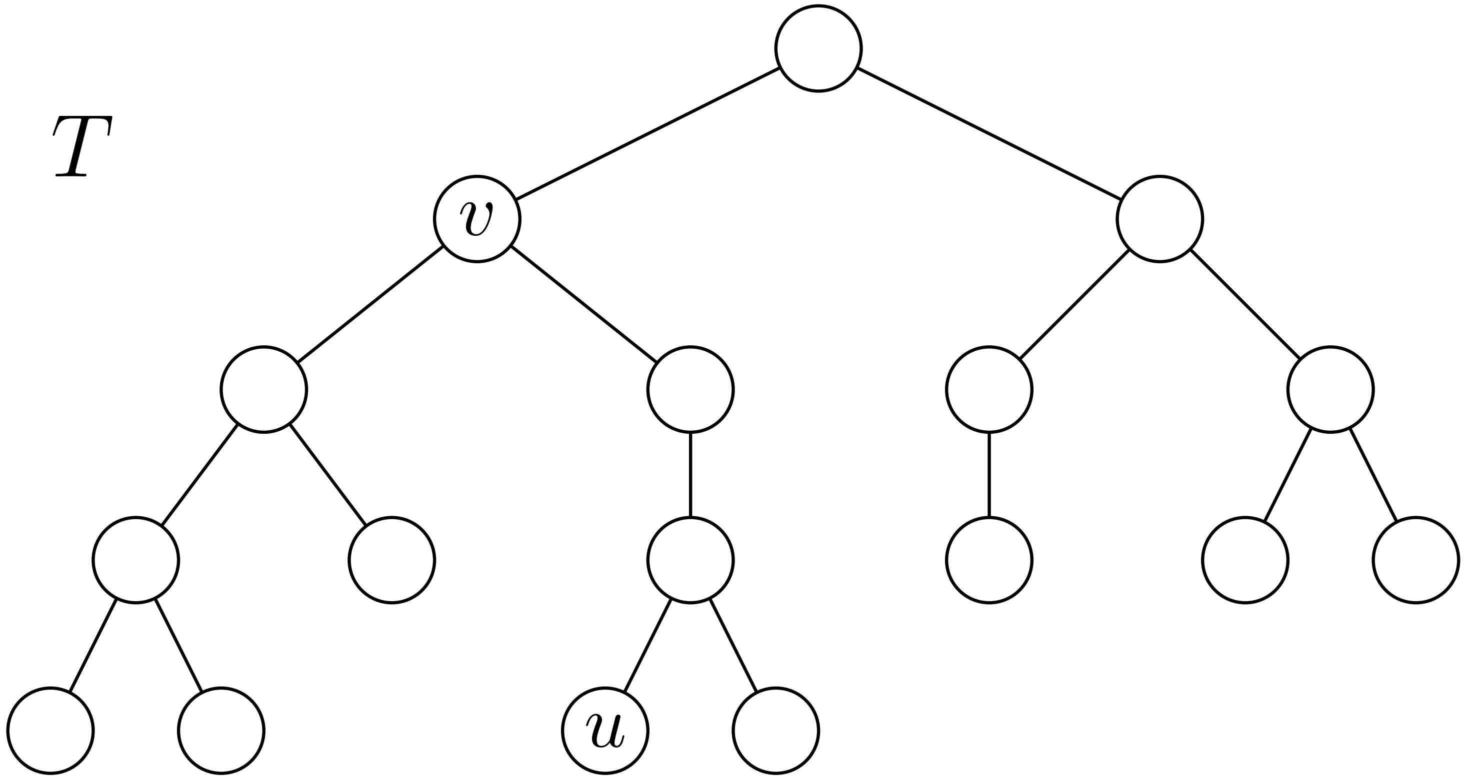
Definition: The *lowest common ancestor* $\text{LCA}_T(u, v)$ of u and v in a rooted tree T is the vertex of maximum depth that is an ancestor of both u and v.

Lowest Common Ancestors



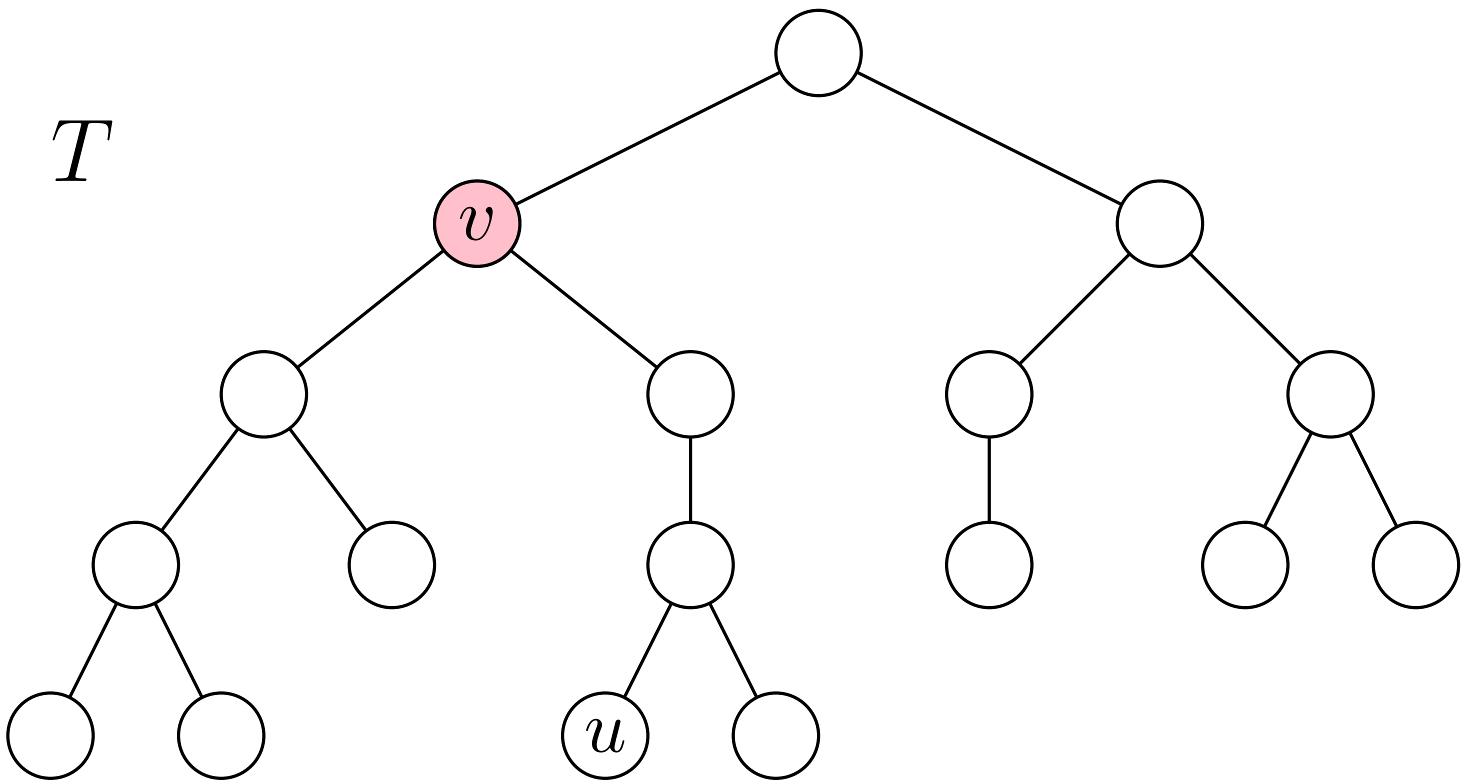
Definition: The *lowest common ancestor* $\mathrm{LCA}_T(u, v)$ of u and v in a rooted tree T is the vertex of maximum depth that is an ancestor of both u and v.

Lowest Common Ancestors



Definition: The *lowest common ancestor* $\mathsf{LCA}_T(u, v)$ of u and v in a rooted tree T is the vertex of maximum depth that is an ancestor of both u and v.

Lowest Common Ancestors



Definition: The *lowest common ancestor* $\mathrm{LCA}_T(u, v)$ of u and v in a rooted tree T is the vertex of maximum depth that is an ancestor of both u and v.

The Problem

Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

The Problem

Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$

The Problem

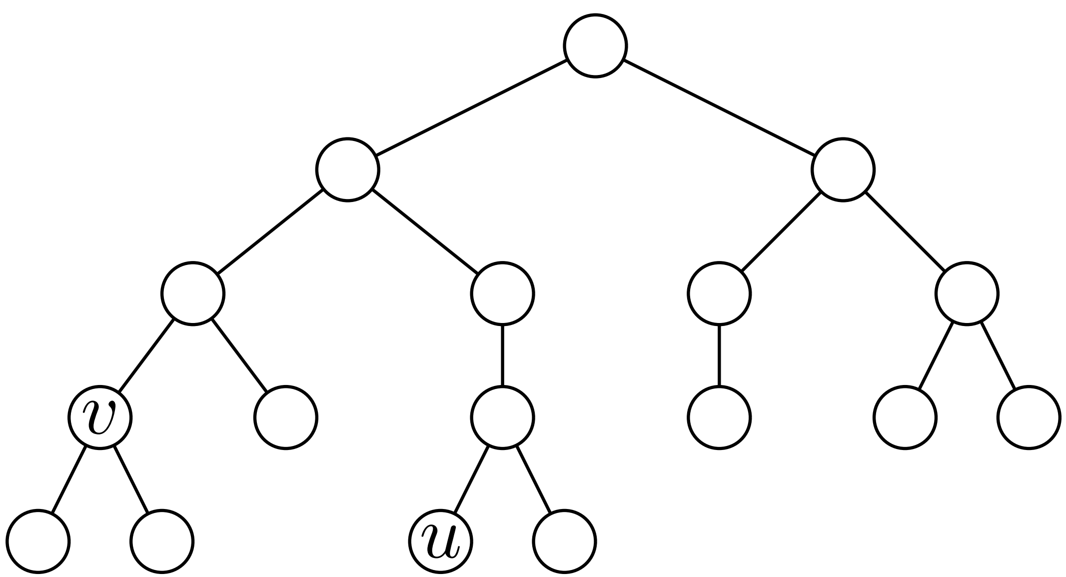
Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(u, v)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



The Problem

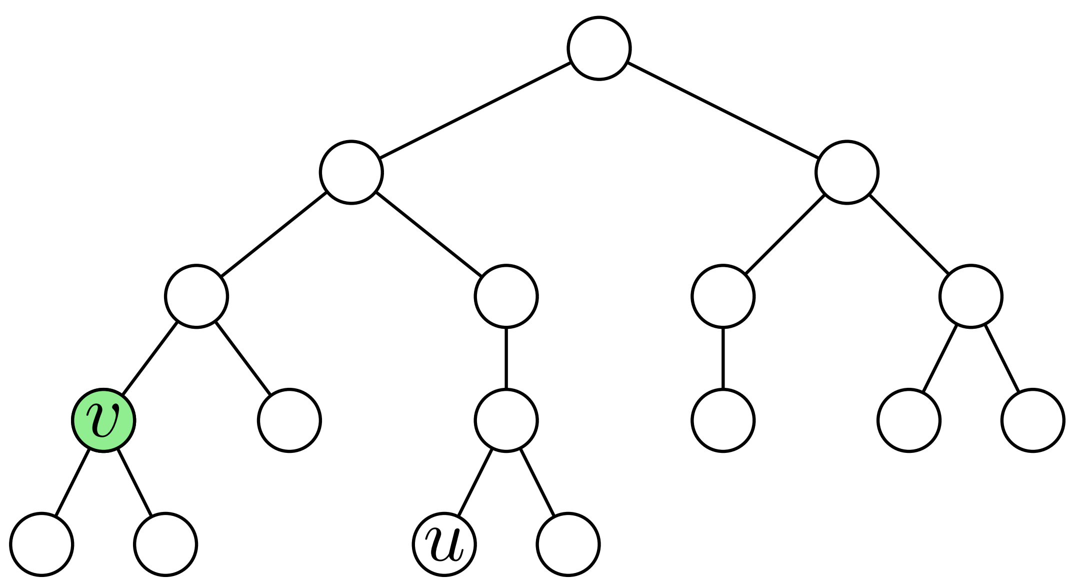
Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none   Size: $O(n)$ Query time: $O(n)$



The Problem

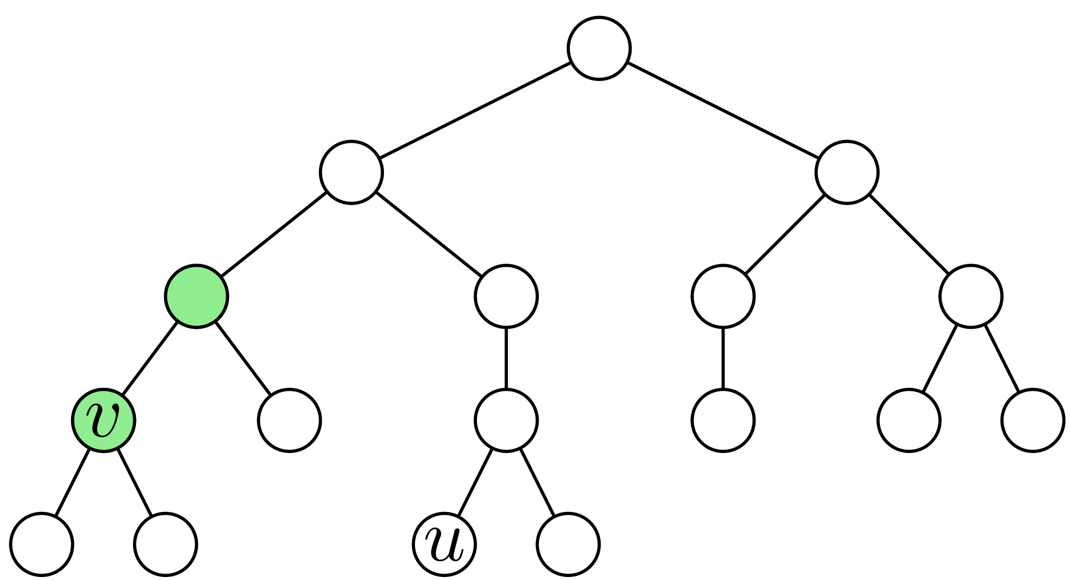
Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none   Size: $O(n)$ Query time: $O(n)$



The Problem

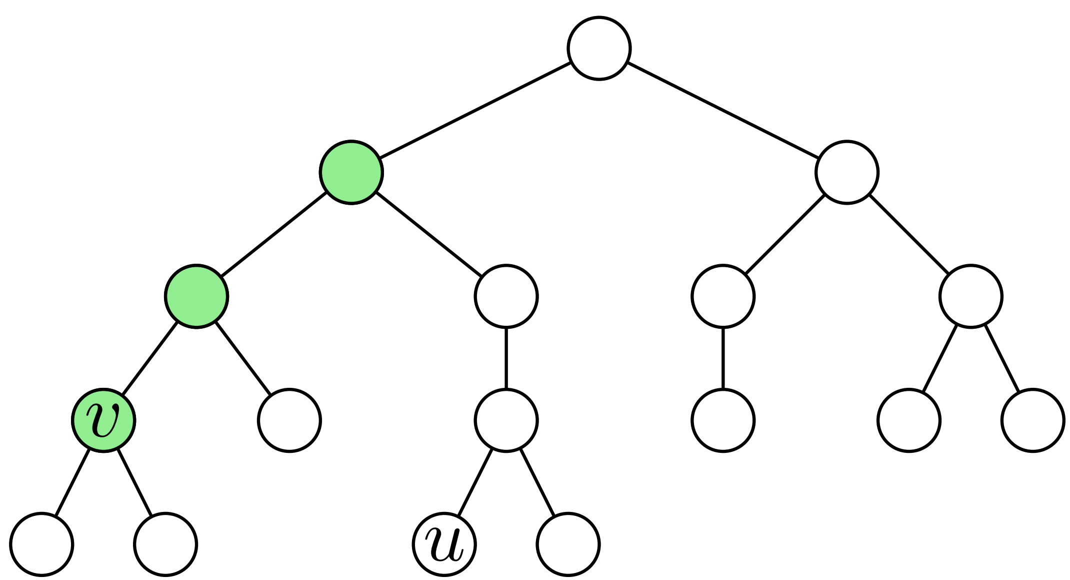
Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(**u, v**)**: report $\mathrm{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



The Problem

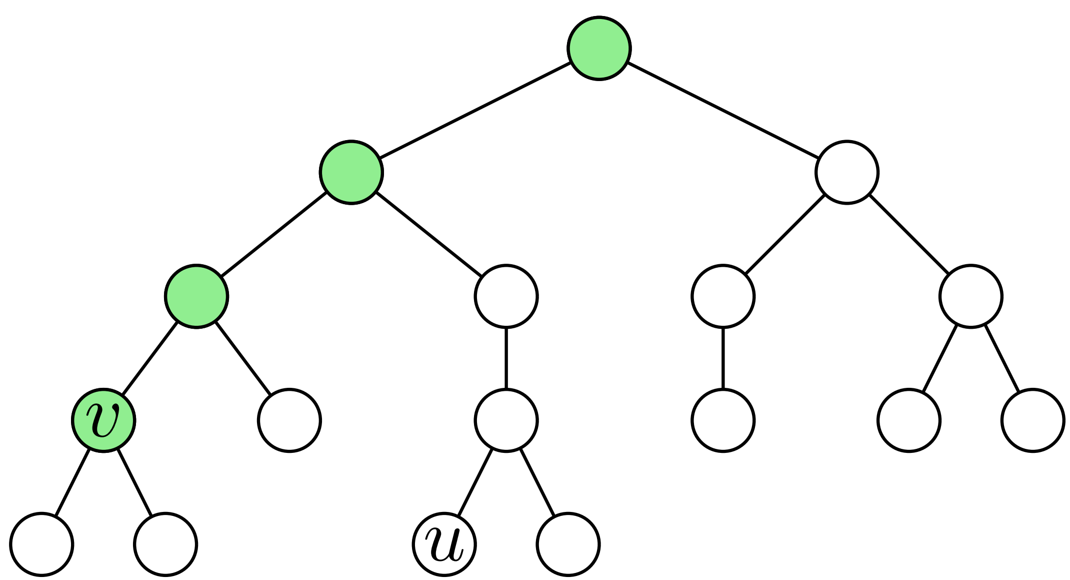
Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(u, v)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



The Problem

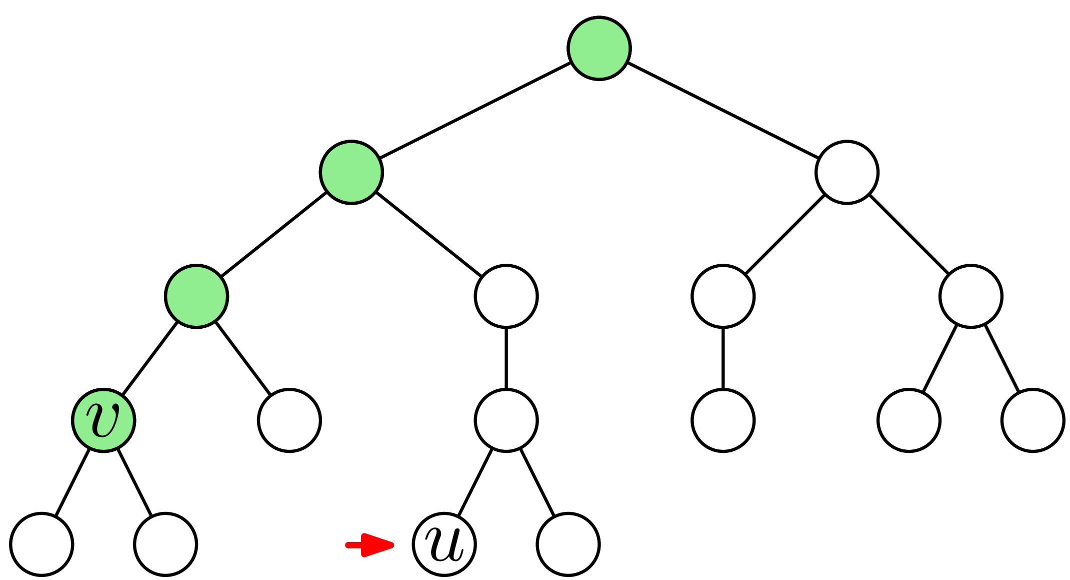
Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



The Problem

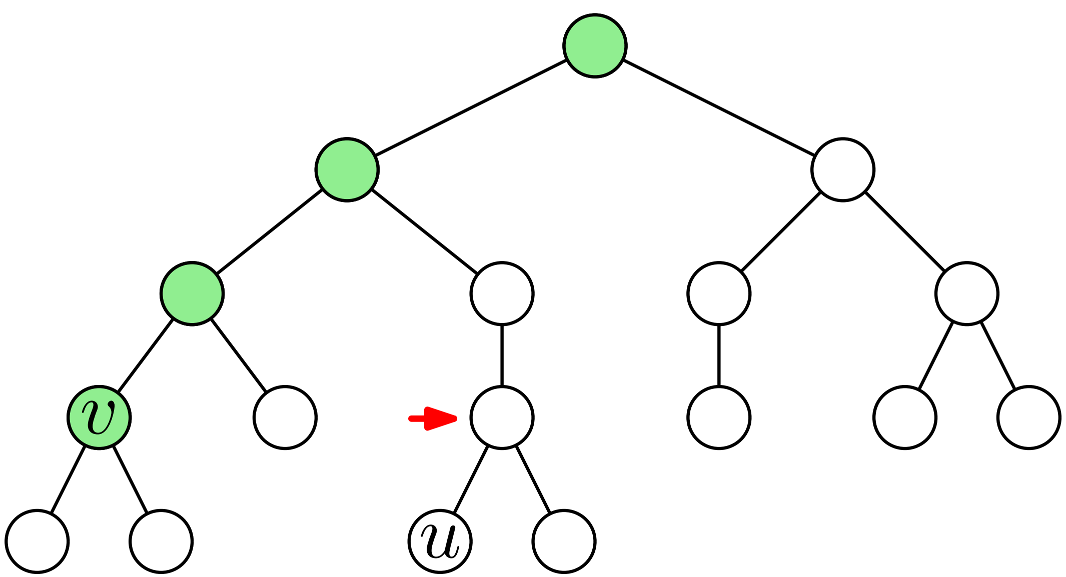
Given T, design a data structure that is able to preprocess T to answer LCA queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



The Problem

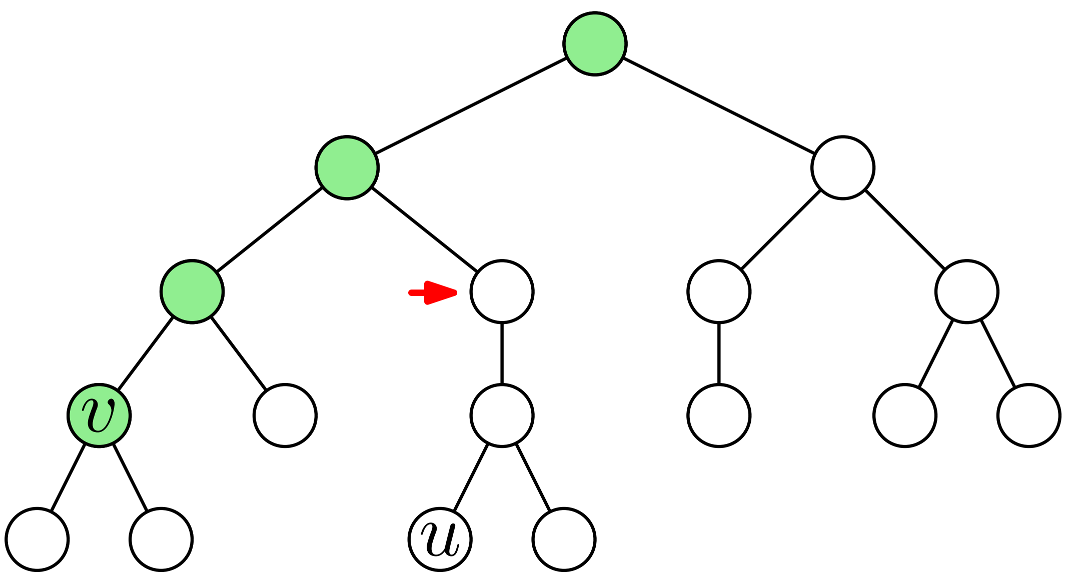
Given T, design a data structure that is able to preprocess T to answer LCA queries:

- **Query(u, v)**: report LCA$_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



The Problem

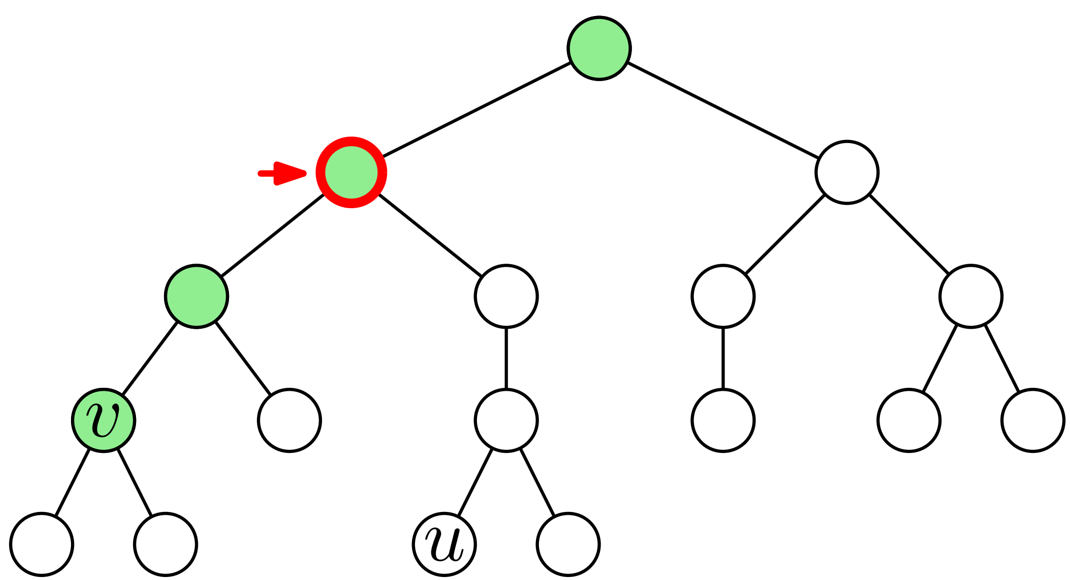
Given T, design a data structure that is able to preprocess T to answer LCA queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



The Problem

Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none   Size: $O(n)$ Query time: $O(n)$
- Preprocessing time: $O(n^3)$ Size: $O(n^2)$ Query time: $O(1)$

The Problem

Given T, design a data structure that is able to preprocess T to answer *LCA* queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$
- Preprocessing time: $O(n^3)$ Size: $O(n^2)$ Query time: $O(1)$

(precompute the answer to all possible queries)

The Problem

Given T, design a data structure that is able to preprocess T to answer LCA queries:

- **Query(**u, v**)**: report $\text{LCA}_T(u, v)$.

Trivial solutions:

$n = \#$ of nodes

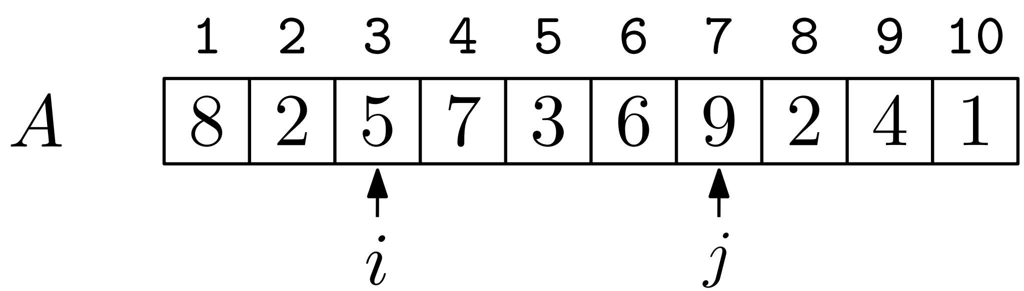
- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$
- Preprocessing time: $O(n^3)$ Size: $O(n^2)$ Query time: $O(1)$
- Preprocessing time: $O(n^2)$ Size: $O(n^2)$ Query time: $O(1)$

$$\text{LCA}_T(u, v) = \begin{cases} \text{LCA}_T(u, v) = u & \text{if } u \text{ is an ancestor of } v \\ \text{LCA}_T(u, v) = \text{LCA}_T(\text{parent}(u), v) & \text{otherwise} \end{cases}$$

A Related Problem

Given an array $A = \langle a_1, \ldots, a_n \rangle$, design a data structure that is able to preprocess A to answer *range minimum* queries:

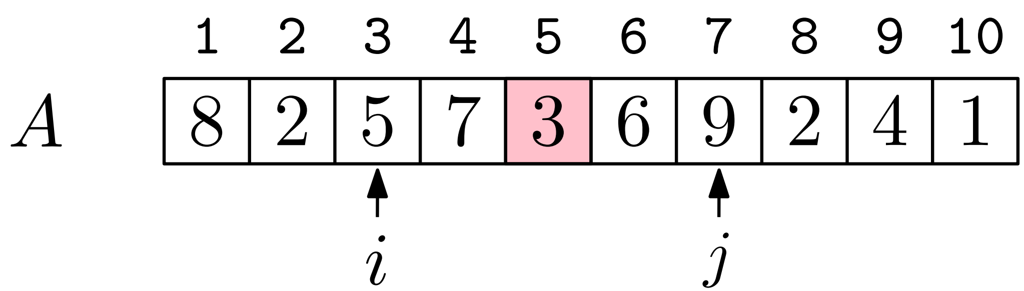
- **RMQ(i, j)**: report an element in $\arg \min_{k=i,\ldots,j} a_k$.



A Related Problem

Given an array $A = \langle a_1, \ldots, a_n \rangle$, design a data structure that is able to preprocess A to answer *range minimum* queries:

- **RMQ(i, j)**: report an element in $\arg \min_{k=i,\ldots,j} a_k$.



RMQ$(3, 7)$=5

A Related Problem

Given an array $A = \langle a_1, \ldots, a_n \rangle$, design a data structure that is able to preprocess A to answer *range minimum* queries:

- **RMQ(i, j)**: report an element in $\arg \min_{k=i,\ldots,j} a_k$.

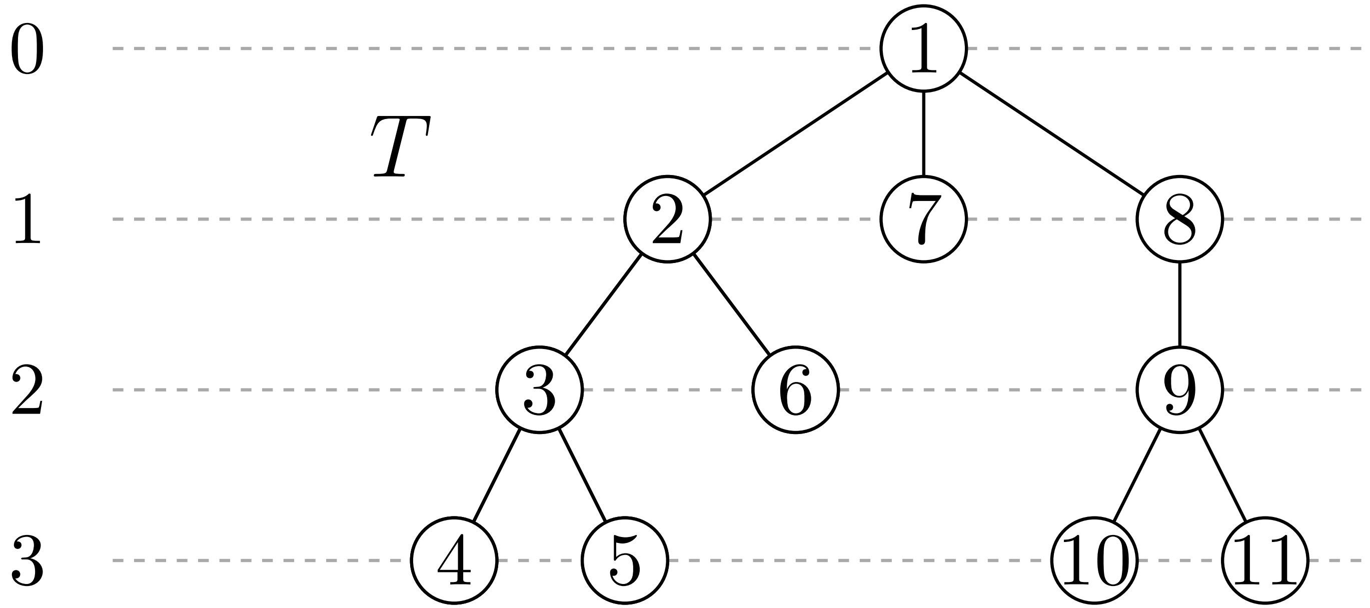
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	3	6	9	2	4	1
			i				j			

RMQ$(3, 7) = 5$

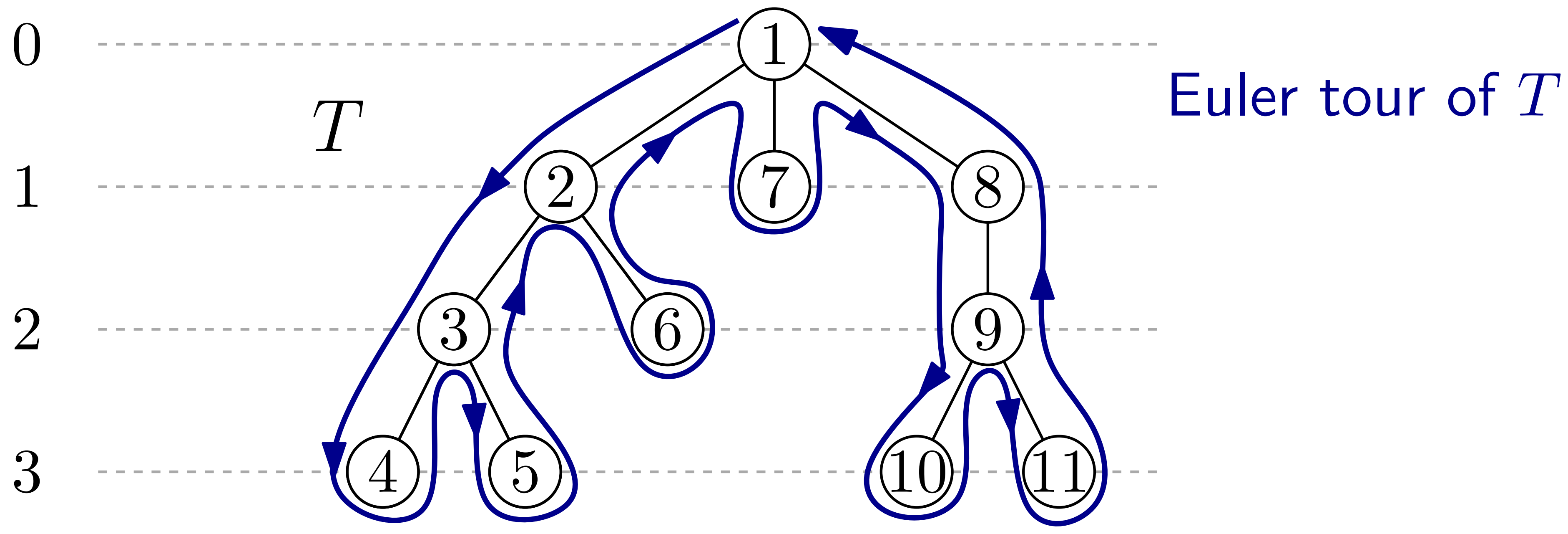
Trivial solutions:

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$
- Preprocessing time: $O(n^3)$ Size: $O(n^2)$ Query time: $O(1)$
- Preprocessing time: $O(n^2)$ Size: $O(n^2)$ Query time: $O(1)$

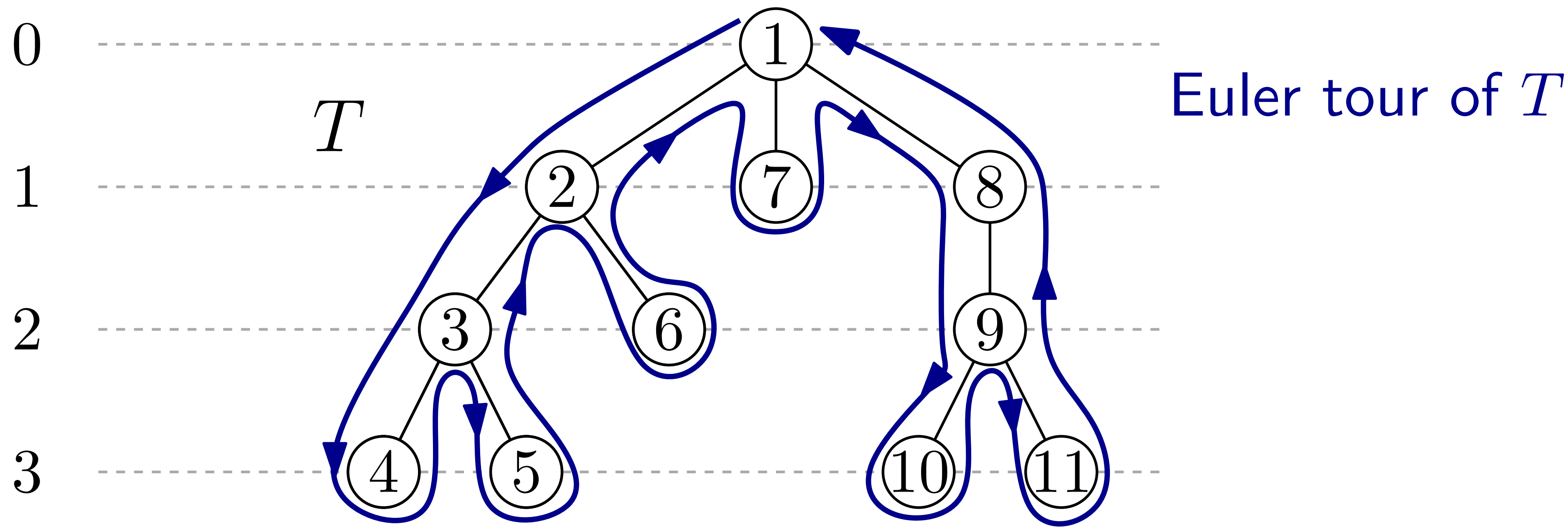
Reducing LCA Queries to RMQ



Reducing LCA Queries to RMQ

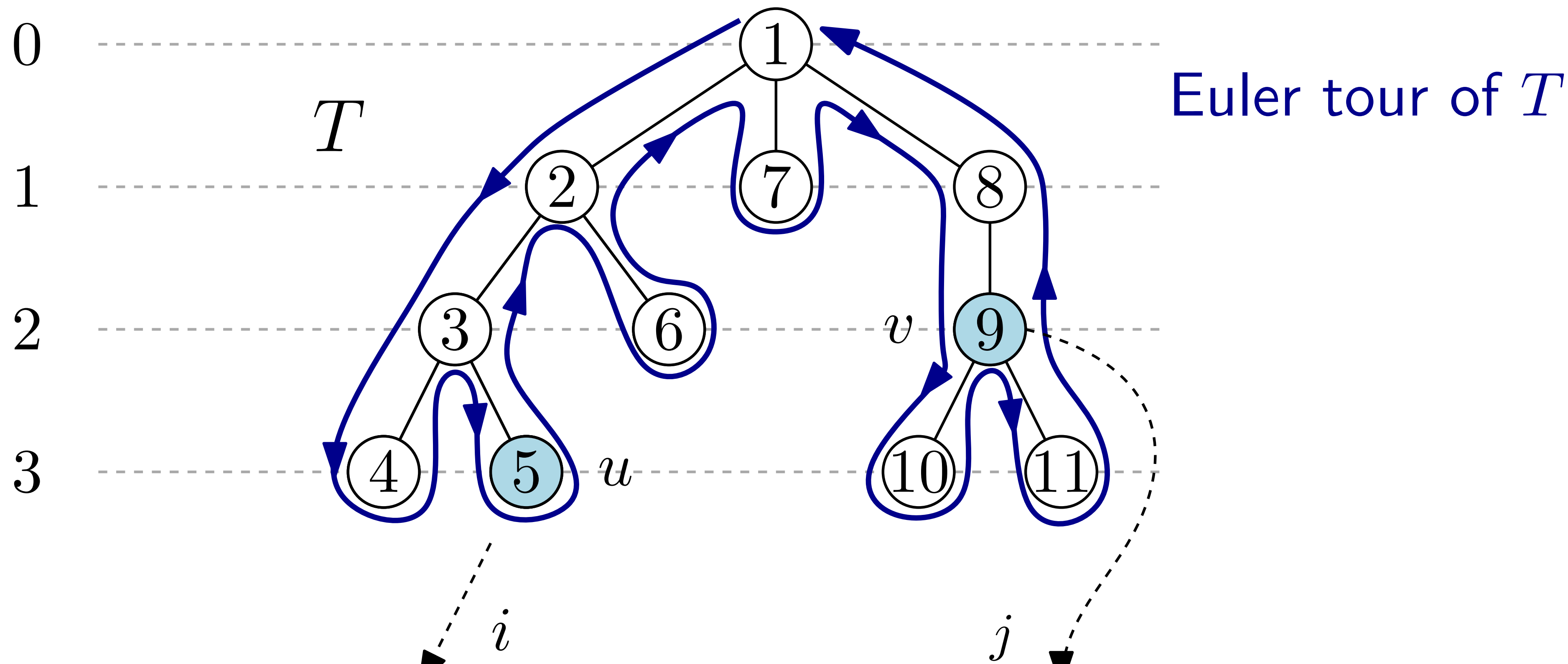


Reducing LCA Queries to RMQ



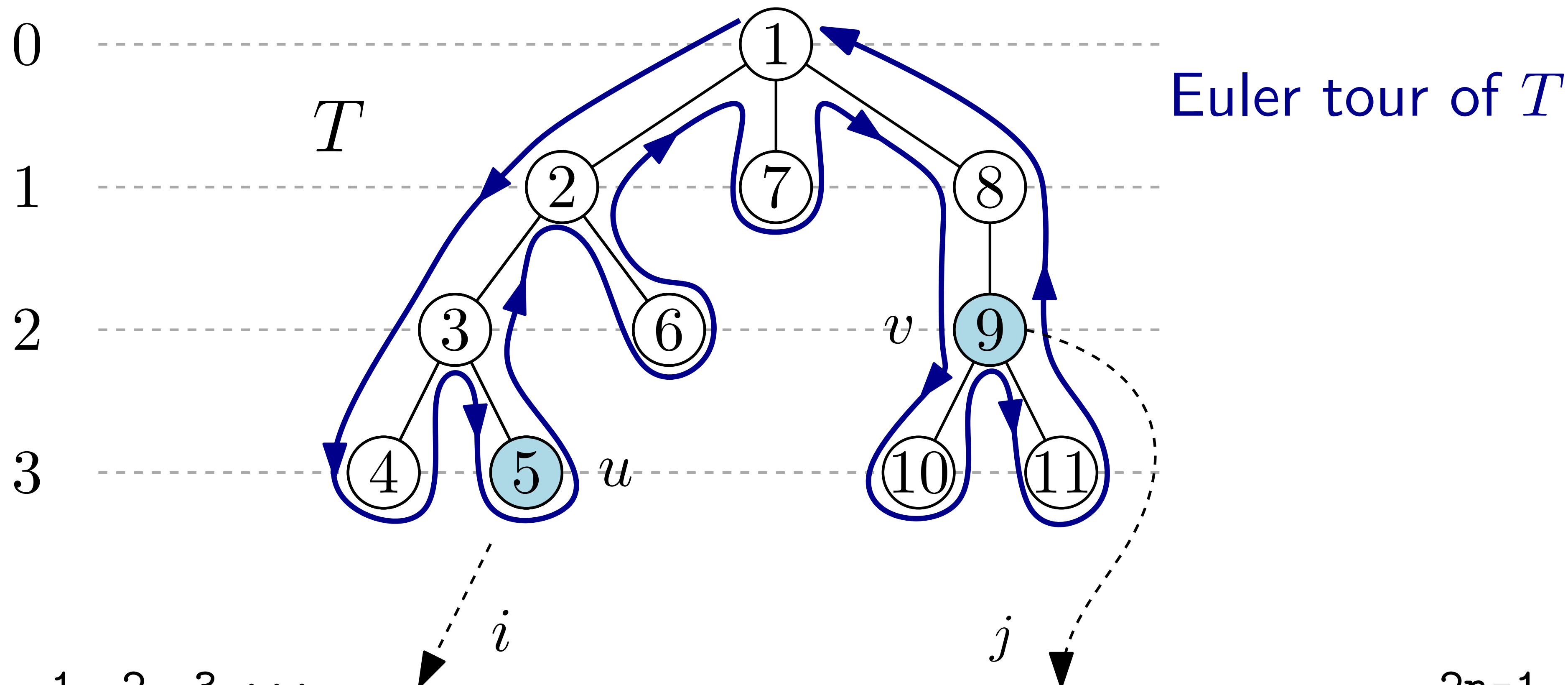
	1	2	3	···																	2n-1
E	1	2	3	4	3	5	3	2	6	2	1	7	1	8	9	10	9	11	9	8	1
D	0	1	2	3	2	3	2	1	2	1	0	1	0	1	2	3	2	3	2	1	0

Reducing LCA Queries to RMQ



	1	2	3	…																	2n-1
E	1	2	3	4	3	5	3	2	6	2	1	7	1	8	9	10	9	11	9	8	1
D	0	1	2	3	2	3	2	1	2	1	0	1	0	1	2	3	2	3	2	1	0

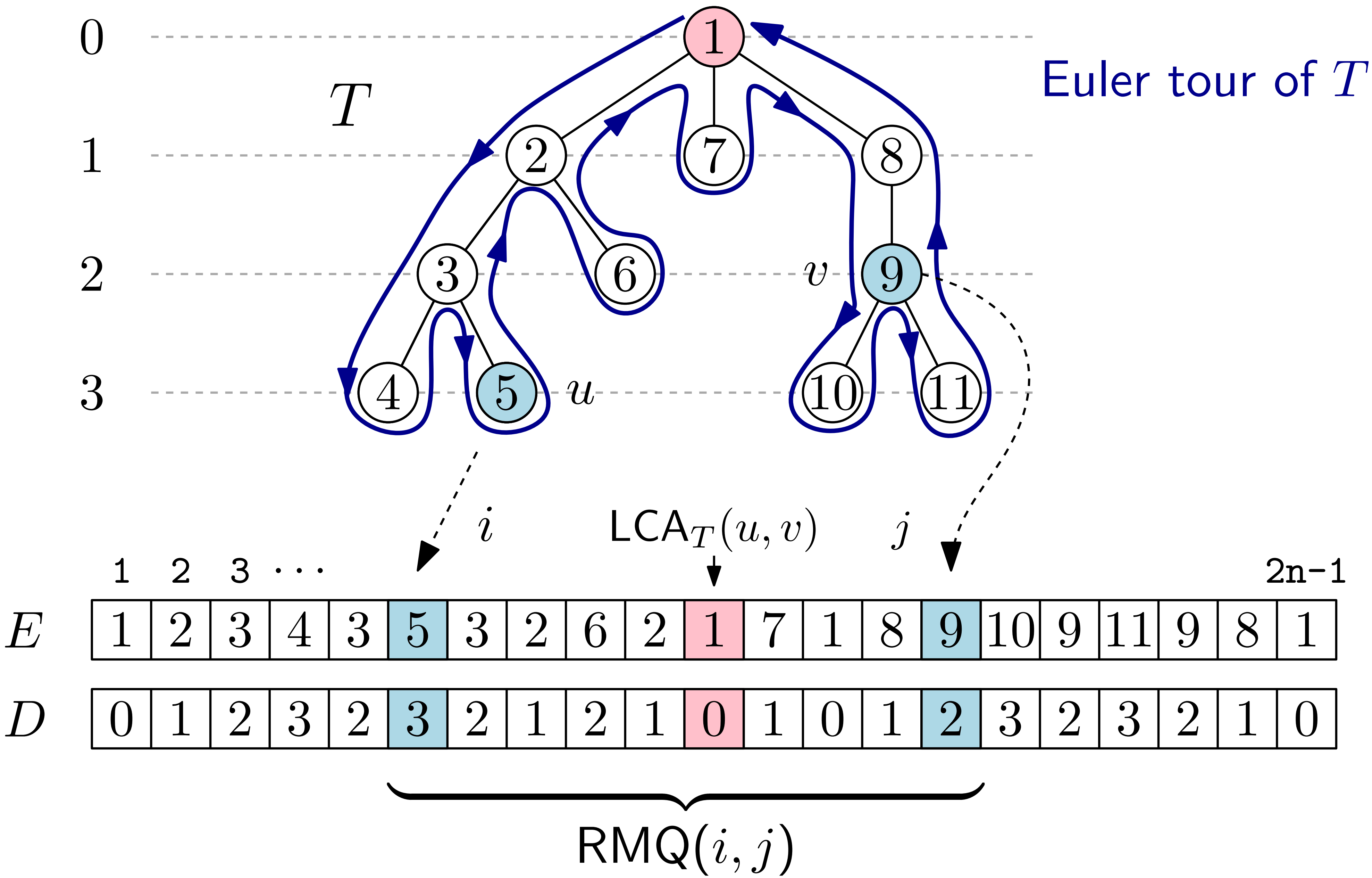
Reducing LCA Queries to RMQ



	1	2	3	...																	2n-1
E	1	2	3	4	3	5	3	2	6	2	1	7	1	8	9	10	9	11	9	8	1
D	0	1	2	3	2	3	2	1	2	1	0	1	0	1	2	3	2	3	2	1	0

RMQ(i, j)

Reducing LCA Queries to RMQ



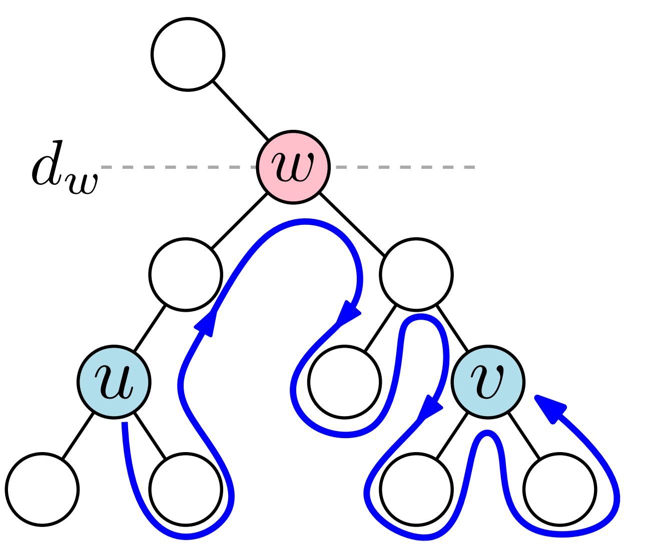
Reducing LCA Queries to RMQ

Let $u, v \in T$ and i (resp. j) be the index of the any occurrence of u (resp. v) in E such that $i \le j$

Claim: $\mathsf{LCA}_T(u, v) = E[\mathsf{RMQ}(i, j)]$

Proof: Let d_w be the depth of $w = \mathsf{LCA}_T(u, v)$ in T

The Euler tour from i to j must pass through w, hence $d_w \in D[i : j]$



Except for w, no other vertex with depth at most d_w appears in the Euler tour from i to j

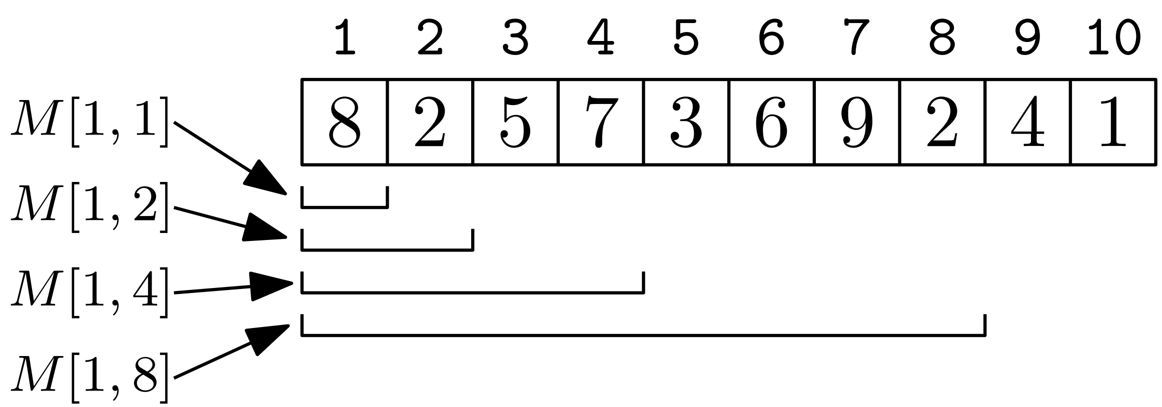
$E[\mathsf{RMQ}(i, j)] = \mathsf{LCA}_T(u, v)$

Solutions to the RMQ problem

"Sparse Table" Solution to RMQ

For $i = 1, \ldots, n$ and $\ell = 2^0, 2^1, \ldots, 2^{\lfloor \log n \rfloor}$, define:

$$M[i, \ell] = \arg \min_{i \leq k < i+\ell} a_k$$



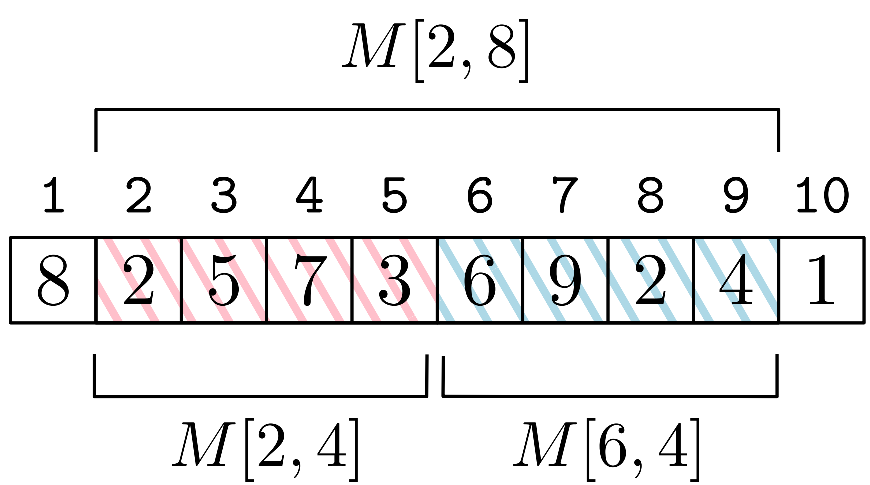
"Sparse Table" Solution to RMQ

For $i = 1, \ldots, n$ and $\ell = 2^0, 2^1, \ldots, 2^{\lfloor \log n \rfloor}$, define:

$$M[i, \ell] = \arg \min_{i \le k < i + \ell} a_k$$

Preprocessing:

$$M[i, \ell] = \begin{cases} i & \text{if } \ell = 1 \\ \arg\min_{k \in \left\{ M\left[i, \frac{\ell}{2}\right], M\left[i + \frac{\ell}{2}, \frac{\ell}{2}\right] \right\}} a_k & \text{if } \ell > 1 \end{cases}$$



“Sparse Table” Solution to RMQ

For $i = 1, \ldots, n$ and $\ell = 2^0, 2^1, \ldots, 2^{\lfloor \log n \rfloor}$, define:

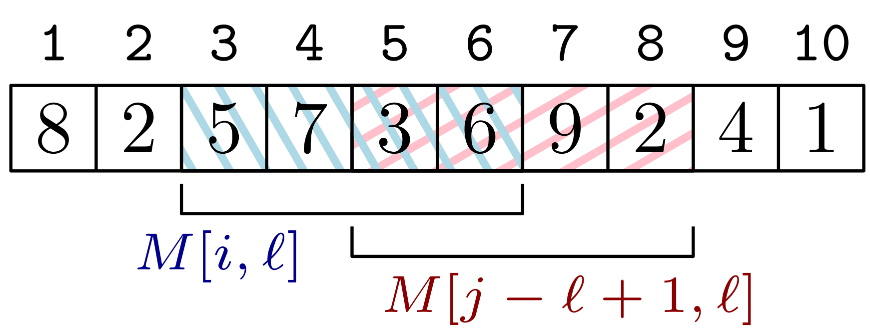
$$M[i, \ell] = \arg \min_{i \le k < i+\ell} a_k$$

Preprocessing:

$$M[i, \ell] = \begin{cases} i & \text{if } \ell = 1 \\ \arg\min_{k \in \left\{ M\left[i, \frac{\ell}{2}\right], M\left[i + \frac{\ell}{2}, \frac{\ell}{2}\right] \right\}} a_k & \text{if } \ell > 1 \end{cases}$$

Answering a query:

Let $\ell = 2^{\lfloor \log(j-i+1) \rfloor}$ 　　 $\text{RMQ}(i, j) = \arg \min_{k \in \{M[i,\ell], M[j-\ell+1,\ell]\}} a_k$



"Sparse Table" Solution to RMQ

For $i = 1, \ldots, n$ and $\ell = 2^0, 2^1, \ldots, 2^{\lfloor \log n \rfloor}$, define:

$$M[i, \ell] = \arg \min_{i \le k < i+\ell} a_k$$

Preprocessing:

$$M[i,\ell] = \begin{cases} i & \text{if } \ell = 1 \\ \arg\min\limits_{k \in \left\{ M\left[i, \frac{\ell}{2}\right], M\left[i + \frac{\ell}{2}, \frac{\ell}{2}\right] \right\}} a_k & \text{if } \ell > 1 \end{cases}$$

Answering a query:

Let $\ell = 2^{\lfloor \log(j-i+1) \rfloor}$ $\qquad$ $\text{RMQ}(i,j) = \arg \min_{k \in \{M[i,\ell], M[j-\ell+1,\ell]\}} a_k$

- Preprocessing time: $O(n \log n)$
- Size: $O(n \log n)$
- Query time: $O(1)$

RMQ Solutions so far

Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	

RMQ Solutions so far

Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table

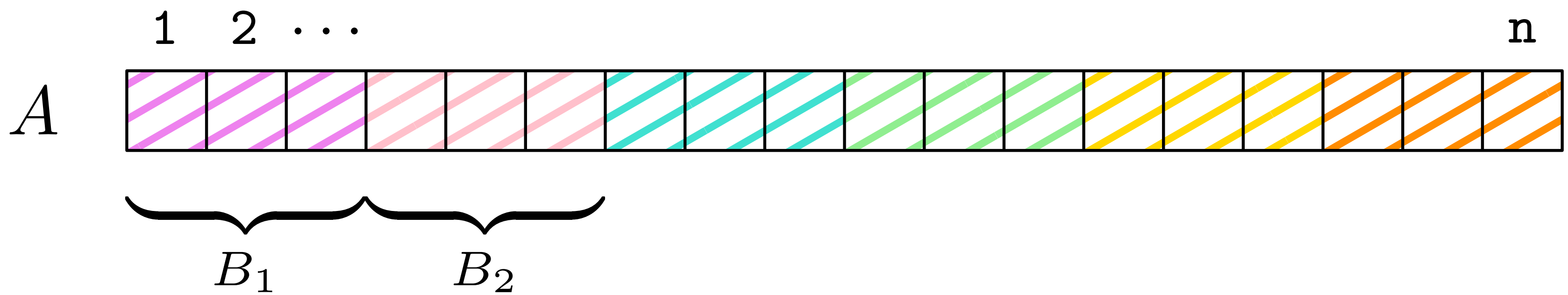
RMQ Solutions so far

Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \underline{\log n})$	$O(n \underline{\log n})$	$O(1)$	Sparse Table

We want to get rid of the $\log n$ factor!

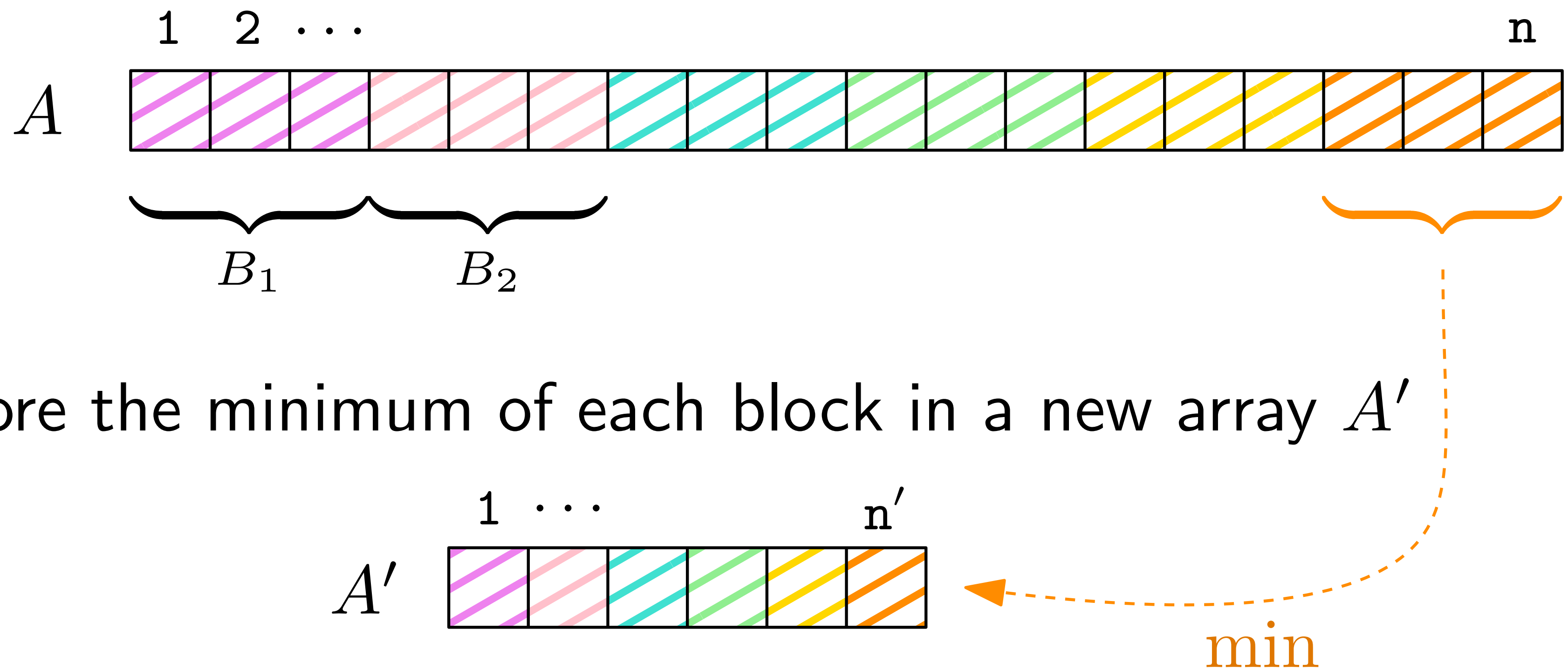
A more compact RMQ oracle

- Logically split A into $\Theta(\frac{n}{\log n})$ "blocks" of $d = \Theta(\log n)$ elements each.



A more compact RMQ oracle

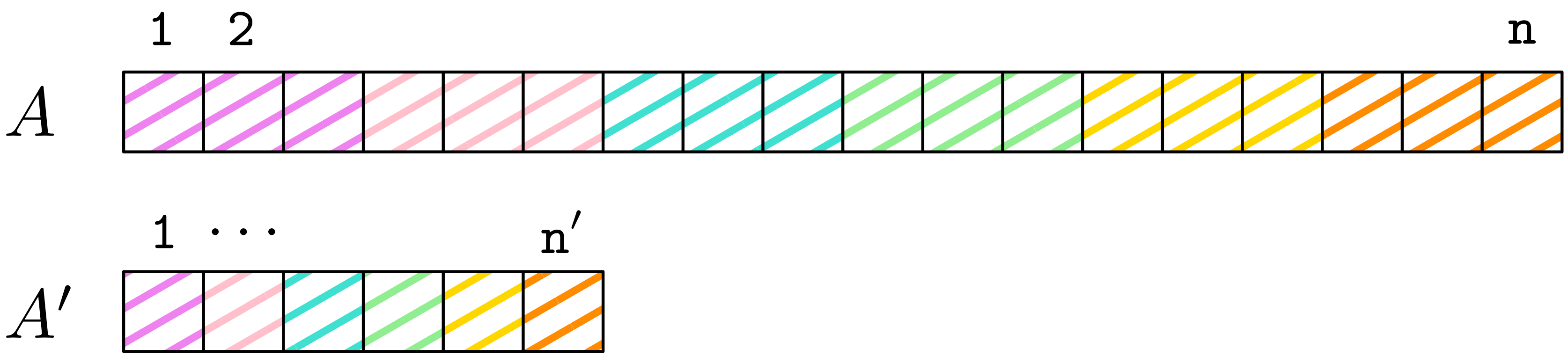
- Logically split A into $\Theta(\frac{n}{\log n})$ "blocks" of $d = \Theta(\log n)$ elements each.



- Store the minimum of each block in a new array A'

Time needed to build A': $O(n)$

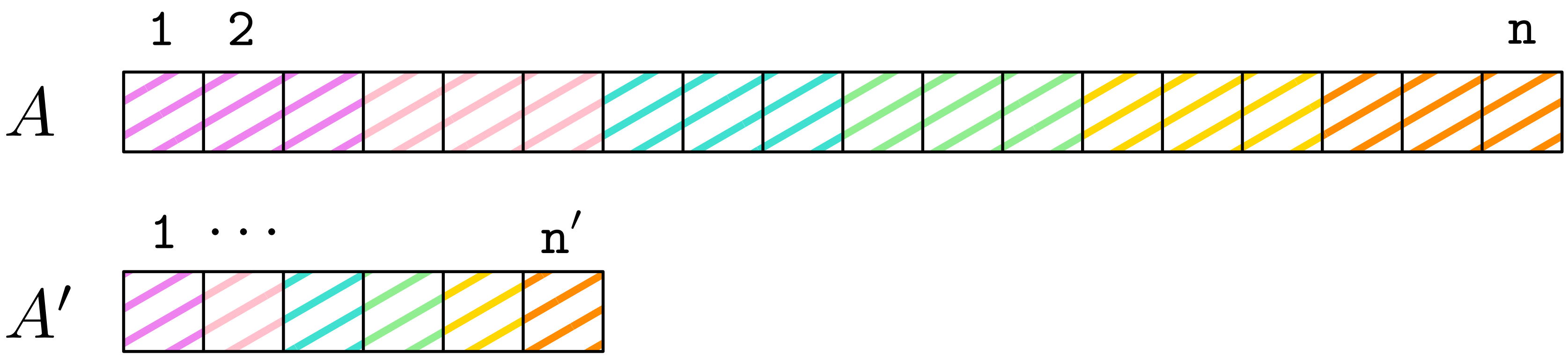
A more compact RMQ oracle



Preprocessing:

- Build the "Sparse Table" oracle $\mathcal{O}$ on A'

A more compact RMQ oracle

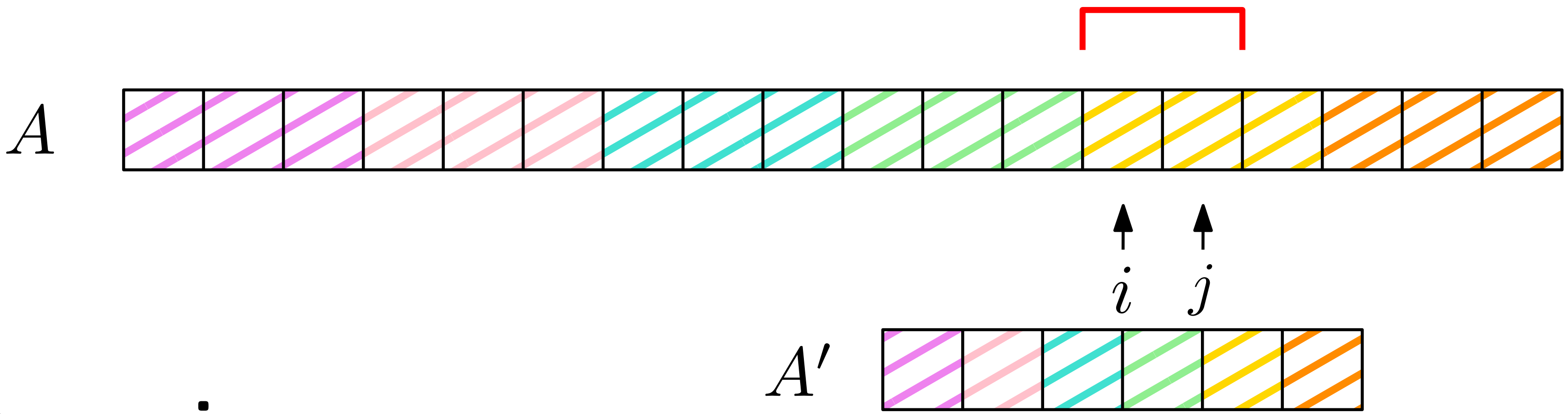


Preprocessing:

- Build the "Sparse Table" oracle $\mathcal{O}$ on A'

Size / time: $O(n' \cdot \log n') = O(\frac{n}{\log n} \cdot \log \frac{n}{\log n}) = O(n)$

A more compact RMQ oracle

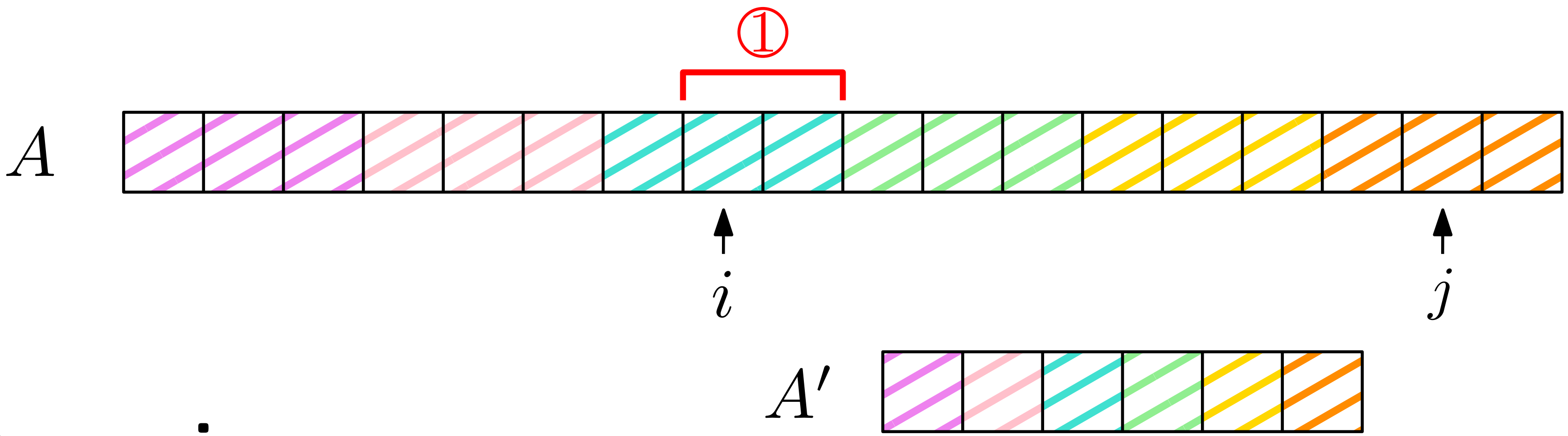


Answering a query:

To answer RMQ(i, j):

- If $i, j \in B_k$ return the position of the minimum in $A[i : j]$

A more compact RMQ oracle



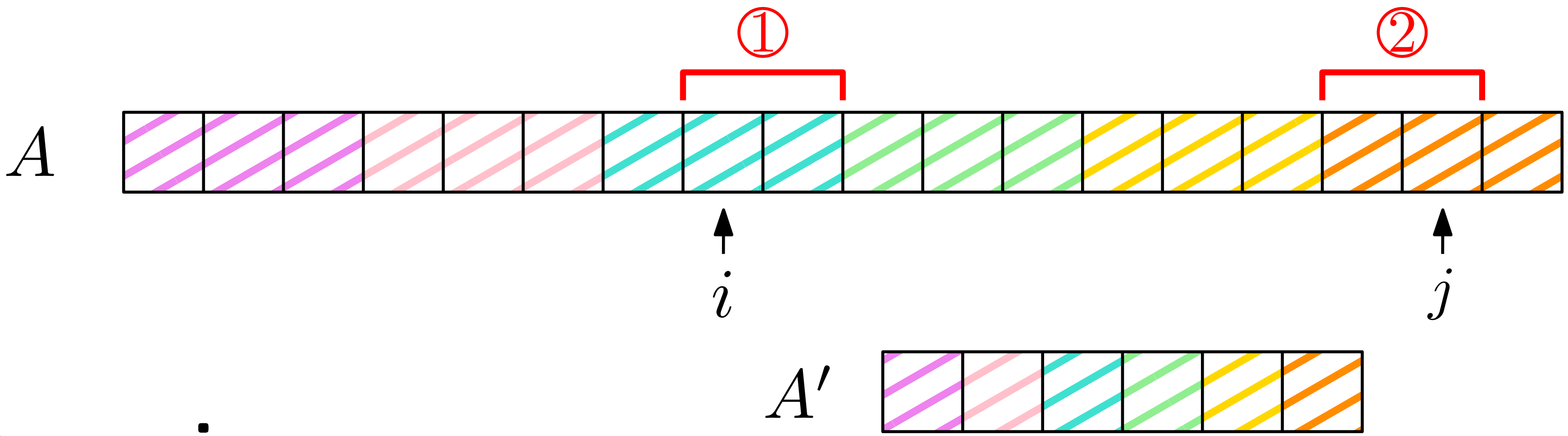
Answering a query:

To answer RMQ(i, j):

- If $i, j \in B_k$ return the position of the minimum in $A[i : j]$
- If $i \in B_h$ and $j \in B_k$, with $k > h$, answer with the position of the smallest element among:

1) The minimum in $A[i : hd]$

A more compact RMQ oracle



Answering a query:

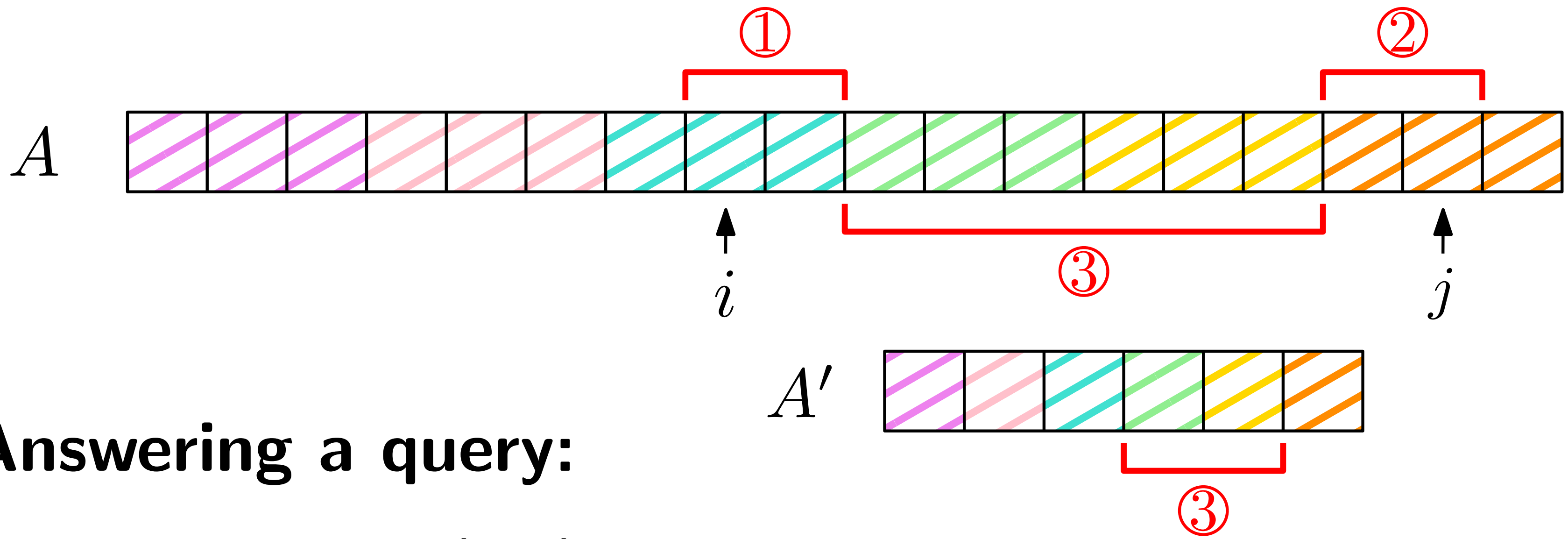
To answer RMQ(i, j):

- If $i, j \in B_k$ return the position of the minimum in $A[i : j]$
- If $i \in B_h$ and $j \in B_k$, with $k > h$, answer with the position of the smallest element among:

1) The minimum in $A[i : hd]$

2) The minimum in $A[(k-1)d + 1 : j]$

A more compact RMQ oracle



Answering a query:

To answer RMQ(i, j):

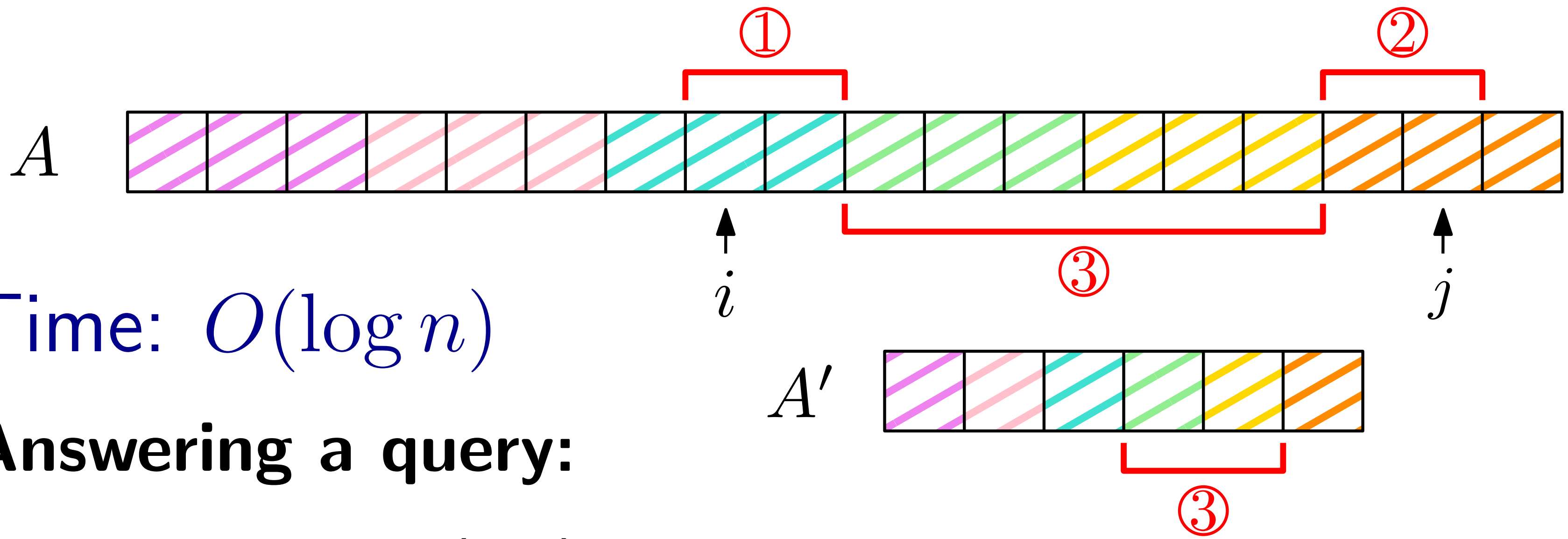
- If $i, j \in B_k$ return the position of the minimum in $A[i : j]$
- If $i \in B_h$ and $j \in B_k$, with $k > h$, answer with the position of the smallest element among:

1) The minimum in $A[i : hd]$

2) The minimum in $A[(k-1)d + 1 : j]$

3) A query to $\mathcal{O}$ to get min $A[hd + 1 : (k-1)d]$

A more compact RMQ oracle



Time: $O(\log n)$

Answering a query:

To answer RMQ(i, j): $O(\log n)$

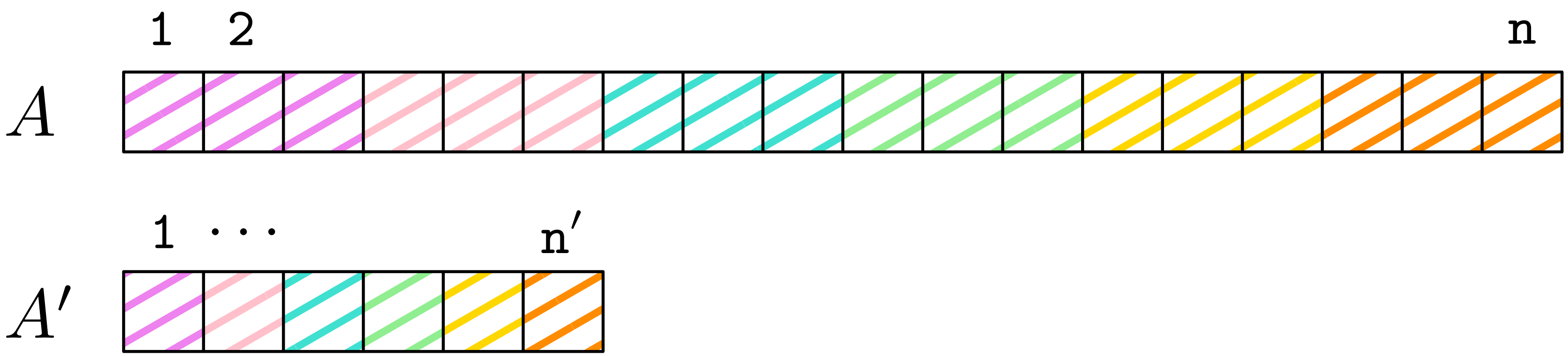
- If $i, j \in B_k$ return the position of the minimum in $A[i : j]$
- If $i \in B_h$ and $j \in B_k$, with $k > h$, answer with the position of the smallest element among:

1) The minimum in $A[i : hd]$ $O(\log n)$

2) The minimum in $A[(k-1)d+1 : j]$ $O(\log n)$

3) A query to $\mathcal{O}$ to get min $A[hd+1 : (k-1)d]$ $O(1)$

A more compact RMQ oracle (alternative)

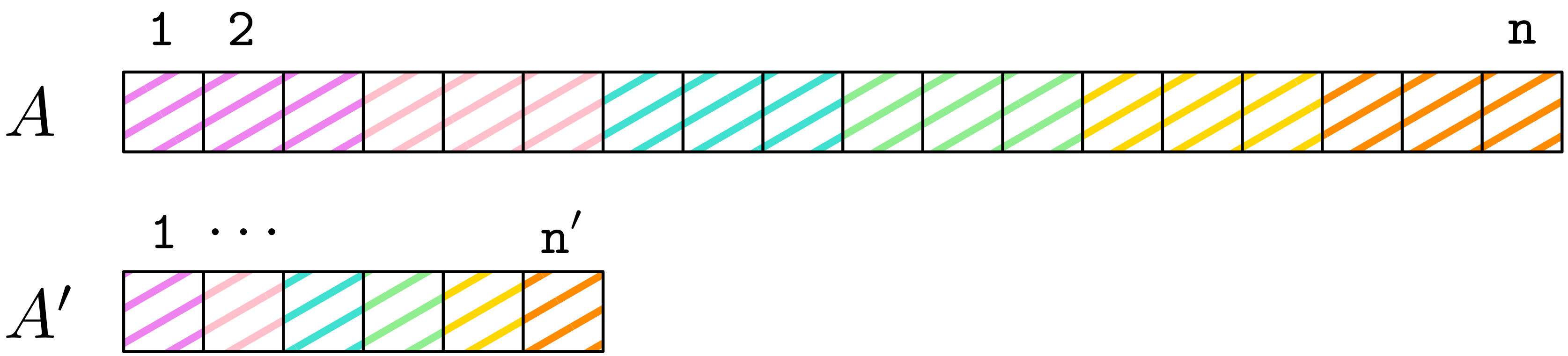


Preprocessing:

- Build the "Sparse Table" oracle $\mathcal{O}$ on A'

Size / time: $O(n' \cdot \log n') = O(\frac{n}{\log n} \cdot \log \frac{n}{\log n}) = O(n)$

A more compact RMQ oracle (alternative)



Preprocessing:

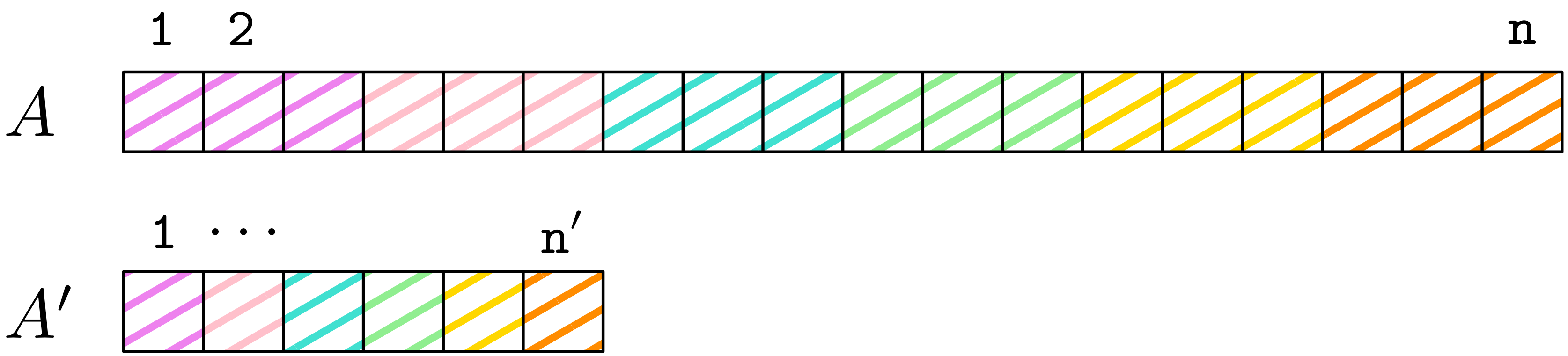
- Build the "Sparse Table" oracle $\mathcal{O}$ on A'

Size / time: $O(n' \cdot \log n') = O(\frac{n}{\log n} \cdot \log \frac{n}{\log n}) = O(n)$

- Build the "Sparse Table" oracle $\mathcal{O}_i$ each B_i

Size / time: $O(\frac{n}{\log n} \cdot (\log n)(\log \log n)) = O(n \log \log n)$

A more compact RMQ oracle (alternative)



Preprocessing:

- Build the "Sparse Table" oracle $\mathcal{O}$ on A'

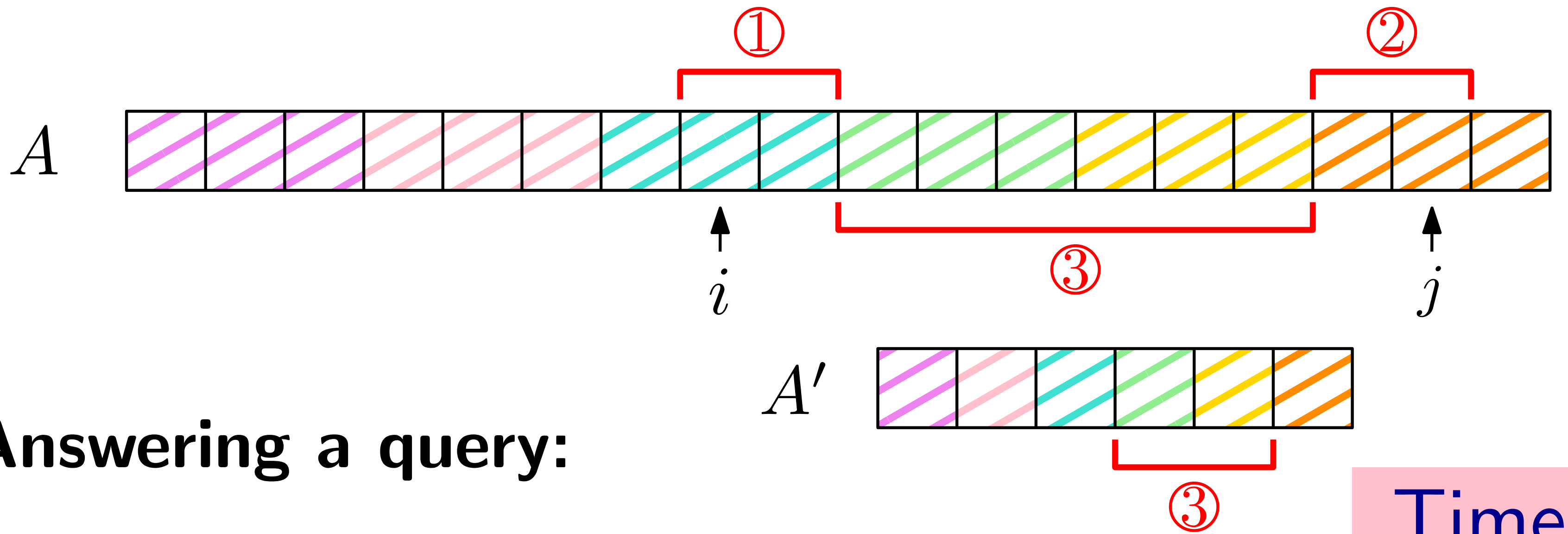
Size / time: $O(n' \cdot \log n') = O(\frac{n}{\log n} \cdot \log \frac{n}{\log n}) = O(n)$

- Build the "Sparse Table" oracle $\mathcal{O}_i$ each B_i

Size / time: $O(\frac{n}{\log n} \cdot (\log n)(\log \log n)) = O(n \log \log n)$

Total size / time: $O(n \log \log n)$

A more compact RMQ oracle



Answering a query:

Time: $O(1)$

To answer RMQ(i, j):

- If i and j are in the same block B_k: query $\mathcal{O}_k$
- If $i \in B_h$ and $j \in B_k$, with $k > h$, answer with the position of the smallest element among those returned by:

1) A query to $\mathcal{O}_h$ to get the minimum in $A[i : hd]$

2) A query to $\mathcal{O}_k$ to get the minimum in $A[(k-1)d + 1 : j]$

3) A query to $\mathcal{O}$ to get the minimum $A[hd + 1 : (k-1)d]$

RMQ Solutions so far

Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table

RMQ Solutions so far

Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table
$O(n)$	$O(n)$	$O(\log n)$	

RMQ Solutions so far

Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table
$O(n)$	$O(n)$	$O(\log n)$	
$O(n \log \log n)$	$O(n \log \log n)$	$O(1)$	

RMQ Solutions so far

Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table
$O(n)$	$O(n)$	$O(\log n)$	
$O(n \log \log n)$	$O(n \log \log n)$	$O(1)$	

Almost...

A Special Case

- Assume that $a_{i+1} - a_i \in \{+1, -1\}$.

A											
	0	1	2	3	2	3	2	1	2	1	0

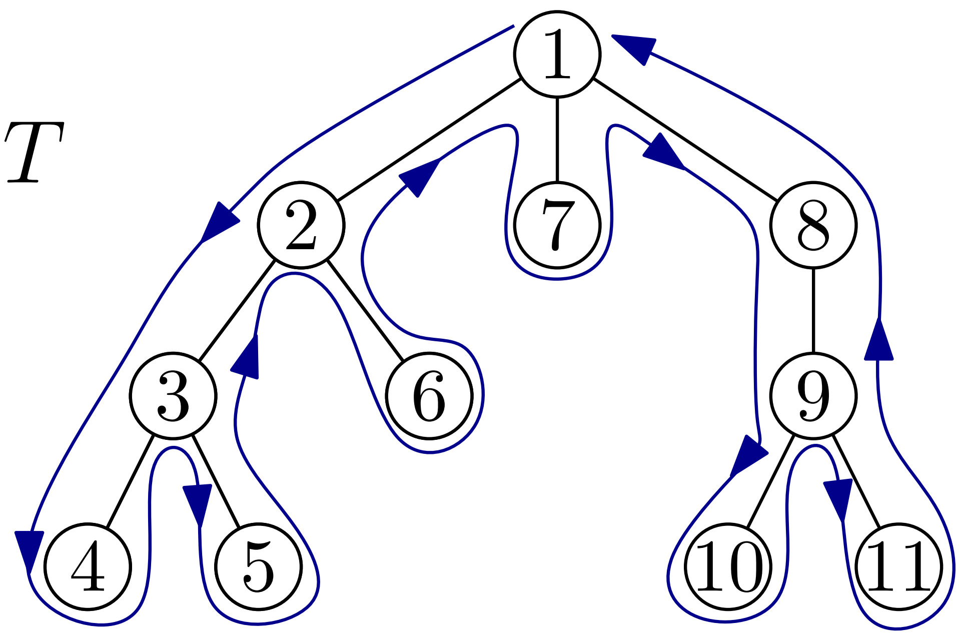
A Special Case

- Assume that $a_{i+1} - a_i \in \{+1, -1\}$.

A	0	1	2	3	2	3	2	1	2	1	0

- This is the case of the instances obtained from LCA !

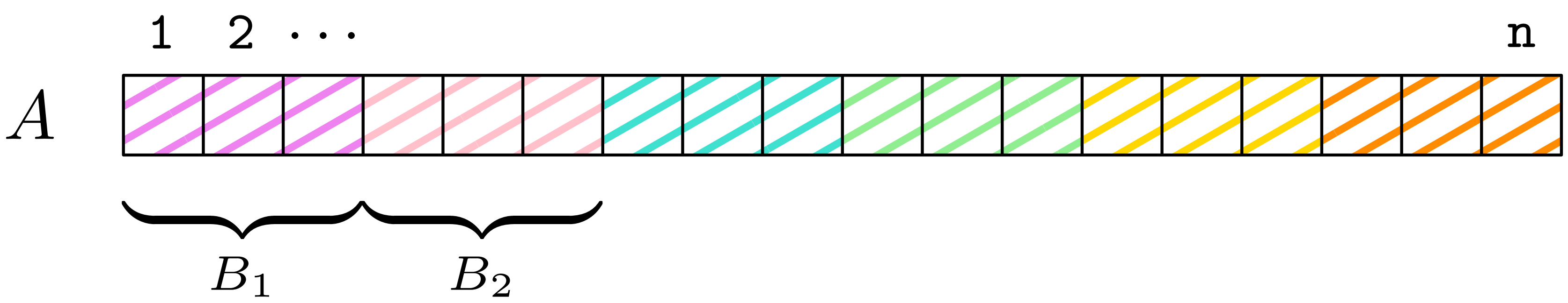
T



	1	2	3	···																	2m-1
E	1	2	3	4	3	5	3	2	6	2	1	7	1	8	9	10	9	11	9	8	1
D	0	1	2	3	2	3	2	1	2	1	0	1	0	1	2	3	2	3	2	1	0

A Special Case

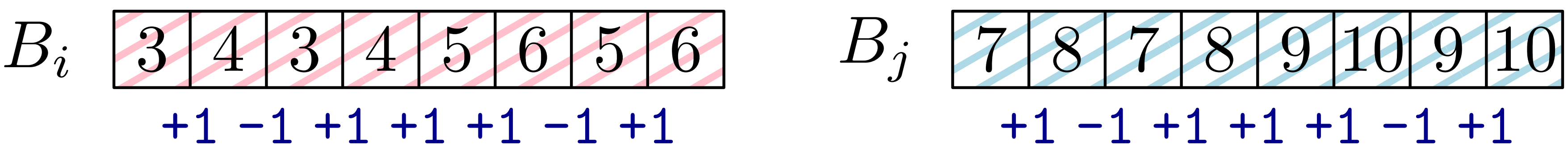
Logically split A into $\Theta(\frac{n}{\log n})$ "blocks" of $d = c \log n$ elements.



A Special Case

Logically split A into $\Theta(\frac{n}{\log n})$ “blocks” of $d = c \log n$ elements.

Definition: Two blocks have the same *type* if they have the same sequence of ± 1 differences between consecutive elements.



A Special Case

Logically split A into $\Theta(\frac{n}{\log n})$ "blocks" of $d = c \log n$ elements.

Definition: Two blocks have the same *type* if they have the same sequence of ± 1 differences between consecutive elements.

B_i							
3	4	3	4	5	6	5	6

+1 −1 +1 +1 +1 −1 +1

B_j							
7	8	7	8	9	10	9	10

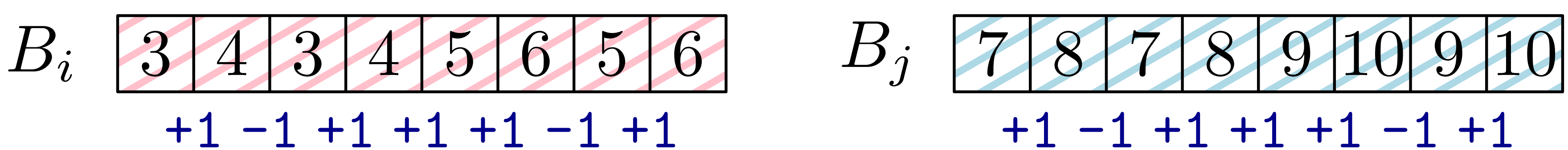
+1 −1 +1 +1 +1 −1 +1

Observation: The answer to the same RMQ query on two blocks of the same type is the same.

A Special Case

Logically split A into $\Theta(\frac{n}{\log n})$ "blocks" of $d = c \log n$ elements.

Definition: Two blocks have the same *type* if they have the same sequence of ± 1 differences between consecutive elements.



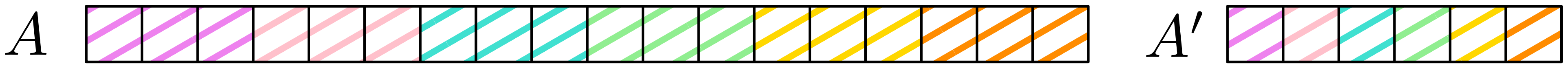
Observation: The answer to the same RMQ query on two blocks of the same type is the same.

How many block types are there?

- Encode a block by its sequence of differences.
- At most $2^{c \log n} = n^c$ block types.

A Special Case

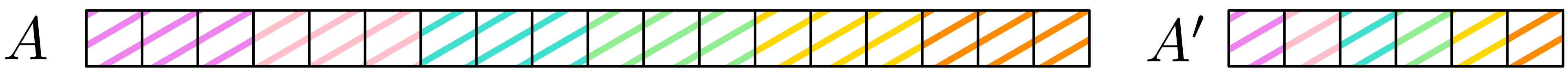
Logically split A into $\Theta(\frac{n}{\log n})$ "blocks" of $d = c \log n$ elements.



- Compute A' and build the "Sparse Table" oracle $\mathcal{O}$ on A'.
  - Size/time: $O(n)$

A Special Case

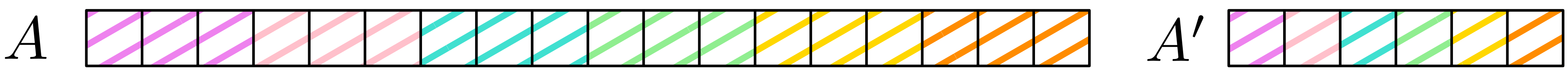
Logically split A into $\Theta(\frac{n}{\log n})$ “blocks” of $d = c \log n$ elements.



- Compute A' and build the “Sparse Table” oracle $\mathcal{O}$ on A'.
  - Size/time: $O(n)$
- For each type t of the at most n^c block types:
  - Build the RMQ oracle $\mathcal{O}_t$ with quadratic preprocessing time/size and constant query time.
  - Size/time: $O(n^c \log^2 n)$

A Special Case

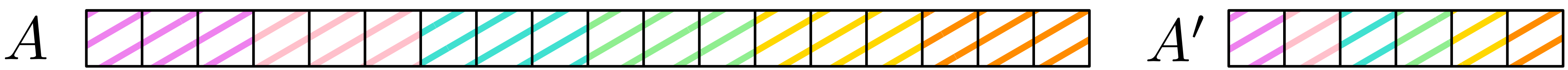
Logically split A into $\Theta(\frac{n}{\log n})$ "blocks" of $d = c \log n$ elements.



- Compute A' and build the "Sparse Table" oracle $\mathcal{O}$ on A'.
  - Size/time: $O(n)$
- For each type t of the at most n^c block types:
  - Build the RMQ oracle $\mathcal{O}_t$ with quadratic preprocessing time/size and constant query time.
  - Size/time: $O(n^c \log^2 n)$
- For each block B_i, store the index t_i of its type.
  - Size/time: $O(\frac{n}{\log n} \cdot \log n^c) = O(n)$.

A Special Case

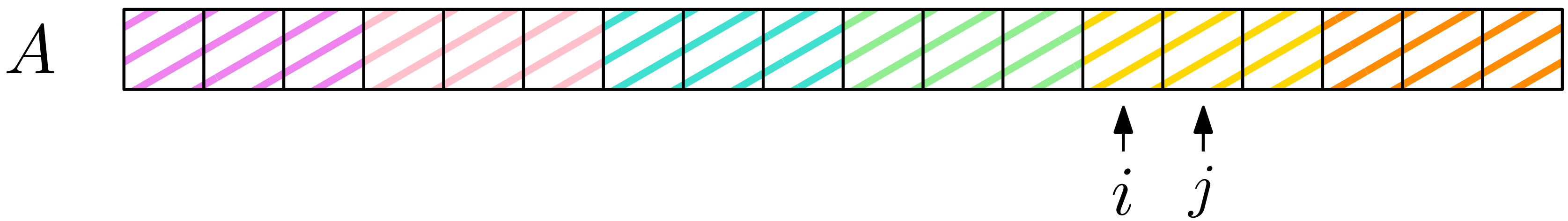
Logically split A into $\Theta(\frac{n}{\log n})$ "blocks" of $d = c \log n$ elements.

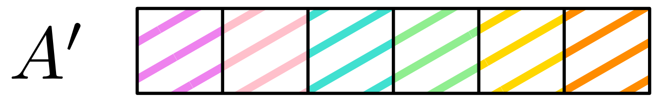


- Compute A' and build the "Sparse Table" oracle $\mathcal{O}$ on A'.
  - Size/time: $O(n)$
- For each type t of the at most n^c block types:
  - Build the RMQ oracle $\mathcal{O}_t$ with quadratic preprocessing time/size and constant query time.
  - Size/time: $O(n^c \log^2 n)$
- For each block B_i, store the index t_i of its type.
  - Size/time: $O(\frac{n}{\log n} \cdot \log n^c) = O(n)$.

Total size/time: $O(n + n^c \log^2 n)$ For (constant) $c < 1$: $O(n)$

A Special Case



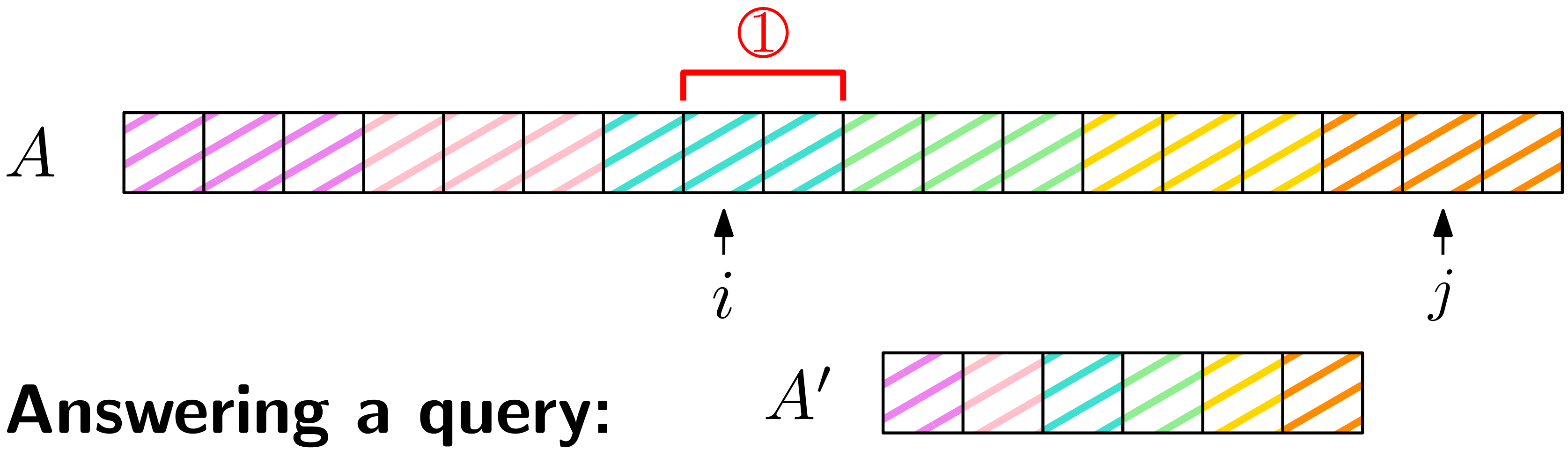


Answering a query:

To answer RMQ(i, j):

- If i and j are in the same block B_k: query $\mathcal{O}_{t_k}$

A Special Case



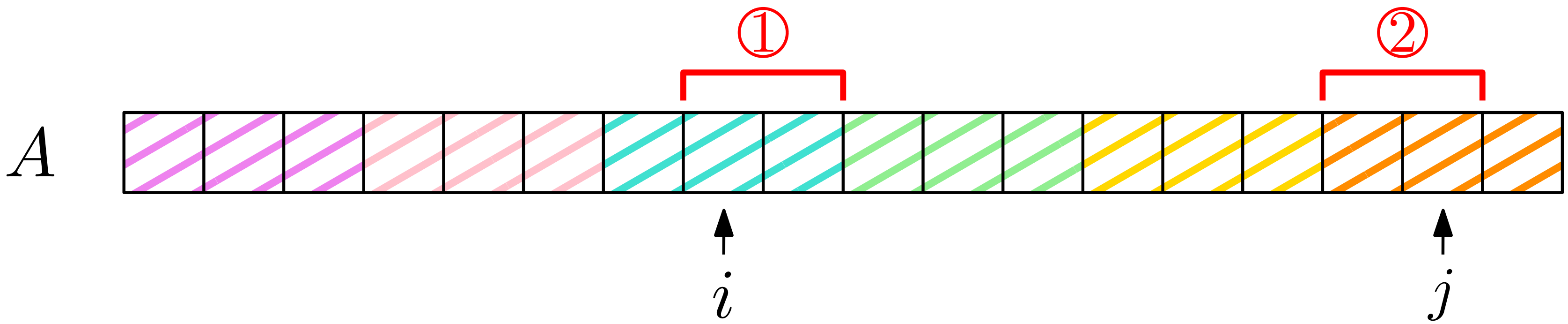
Answering a query:

To answer RMQ(i, j):

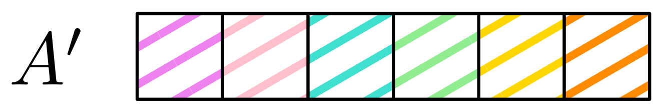
- If i and j are in the same block B_k: query $\mathcal{O}_{t_k}$
- If $i \in B_h$ and $j \in B_k$, with $k > h$, answer with the position of the smallest element among those returned by:

1) A query to $\mathcal{O}_{t_h}$ to get the minimum in $A[i : hd]$

A Special Case



Answering a query:



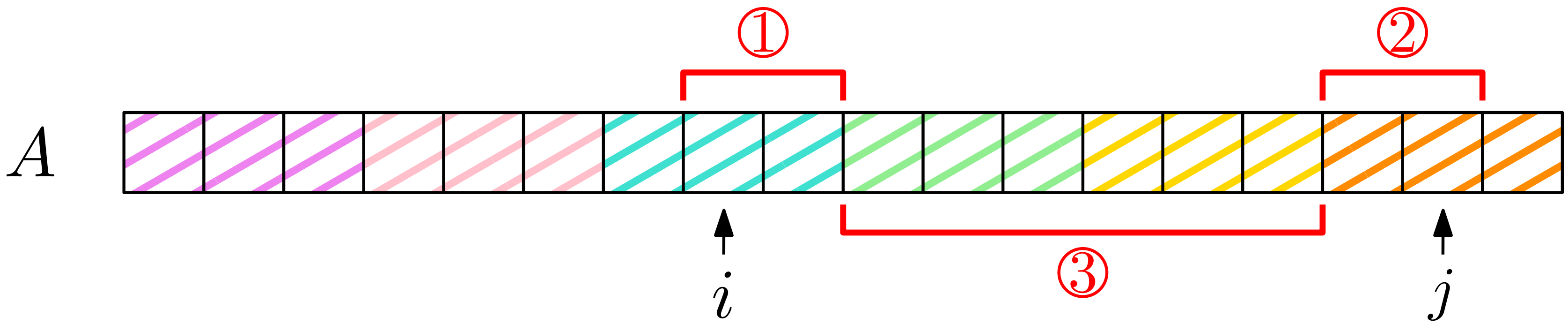
To answer RMQ(i, j):

- If i and j are in the same block B_k: query $\mathcal{O}_{t_k}$
- If $i \in B_h$ and $j \in B_k$, with $k > h$, answer with the position of the smallest element among those returned by:

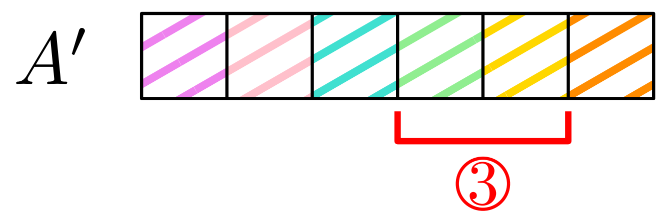
1) A query to $\mathcal{O}_{t_h}$ to get the minimum in $A[i : hd]$

2) A query to $\mathcal{O}_{t_k}$ to get the minimum in $A[(k-1)d + 1 : j]$

A Special Case



Answering a query:



To answer RMQ(i, j):

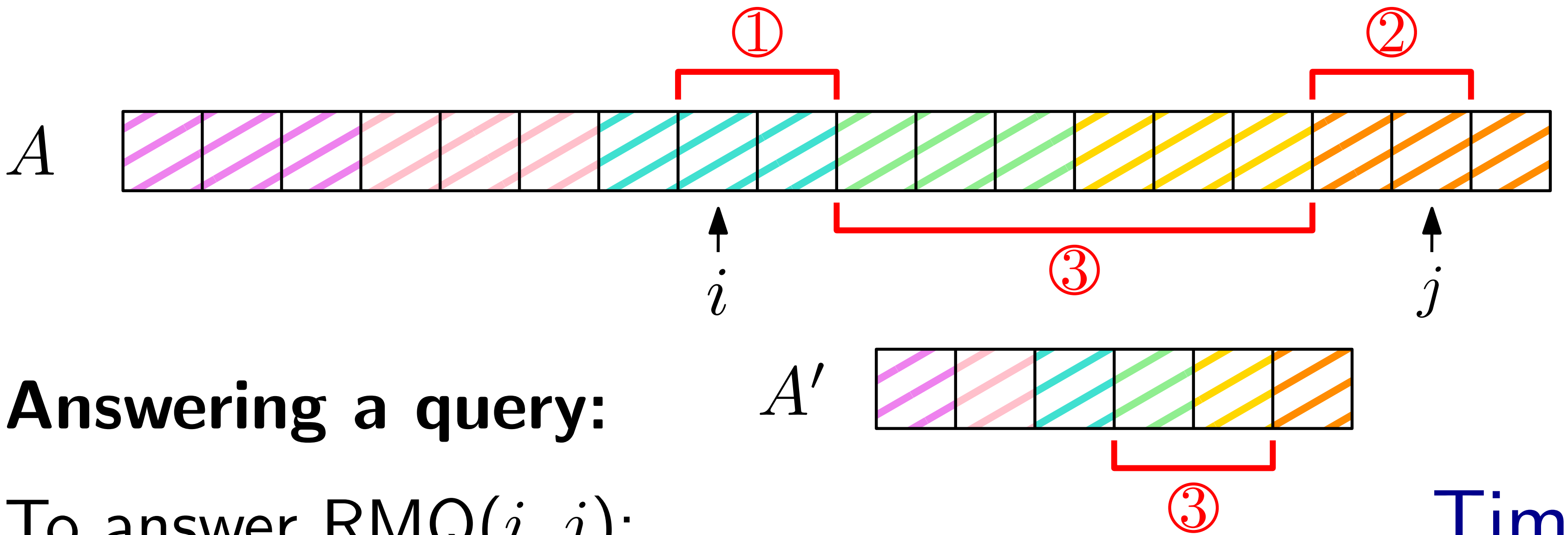
- If i and j are in the same block B_k: query $\mathcal{O}_{t_k}$
- If $i \in B_h$ and $j \in B_k$, with $k > h$, answer with the position of the smallest element among those returned by:

1) A query to $\mathcal{O}_{t_h}$ to get the minimum in $A[i : hd]$

2) A query to $\mathcal{O}_{t_k}$ to get the minimum in $A[(k-1)d + 1 : j]$

3) A query to $\mathcal{O}$ to get the minimum $A[hd + 1 : (k-1)d]$

A Special Case



Answering a query:

To answer RMQ(i, j):

Time: $O(1)$

- If i and j are in the same block B_k: query $\mathcal{O}_{t_k}$
- If $i \in B_h$ and $j \in B_k$, with $k > h$, answer with the position of the smallest element among those returned by:

1) A query to $\mathcal{O}_{t_h}$ to get the minimum in $A[i : hd]$

2) A query to $\mathcal{O}_{t_k}$ to get the minimum in $A[(k-1)d+1 : j]$

3) A query to $\mathcal{O}$ to get the minimum $A[hd+1 : (k-1)d]$

RMQ Solutions so far

Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table
$O(n)$	$O(n)$	$O(\log n)$	
$O(n \log \log n)$	$O(n \log \log n)$	$O(1)$	

RMQ Solutions so far

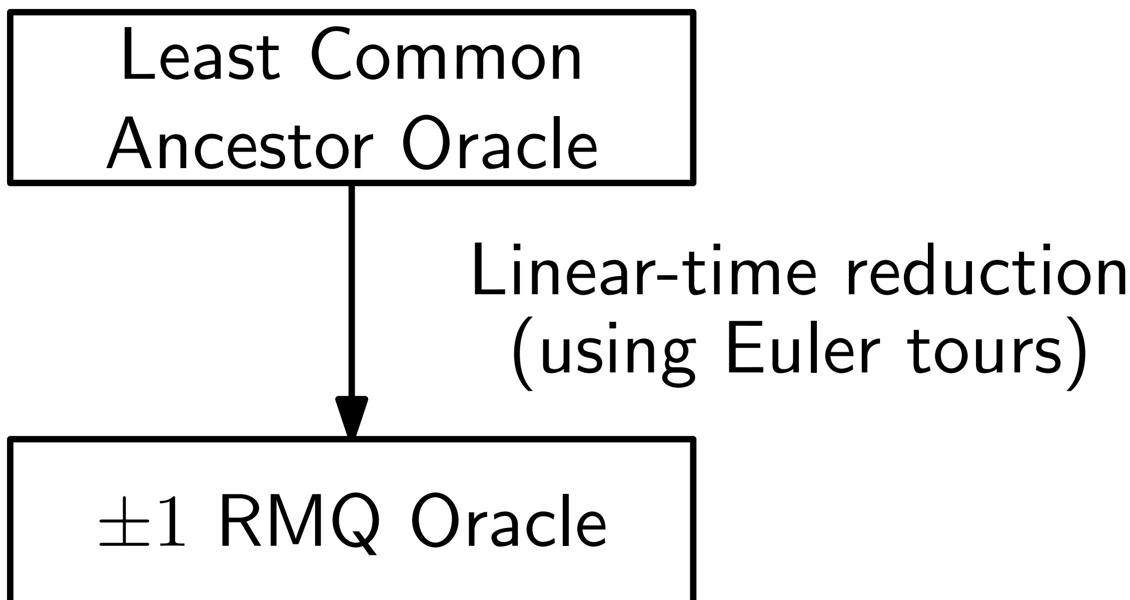
Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table
$O(n)$	$O(n)$	$O(\log n)$	
$O(n \log \log n)$	$O(n \log \log n)$	$O(1)$	
$O(n)$	$O(n)$	$O(1)$	± 1 RMQ

RMQ Solutions so far

Size	Preprocessing Time	Query Time	Notes
$O(n)$	–	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table
$O(n)$	$O(n)$	$O(\log n)$	
$O(n \log \log n)$	$O(n \log \log n)$	$O(1)$	
$O(n)$	$O(n)$	$O(1)$	± 1 RMQ

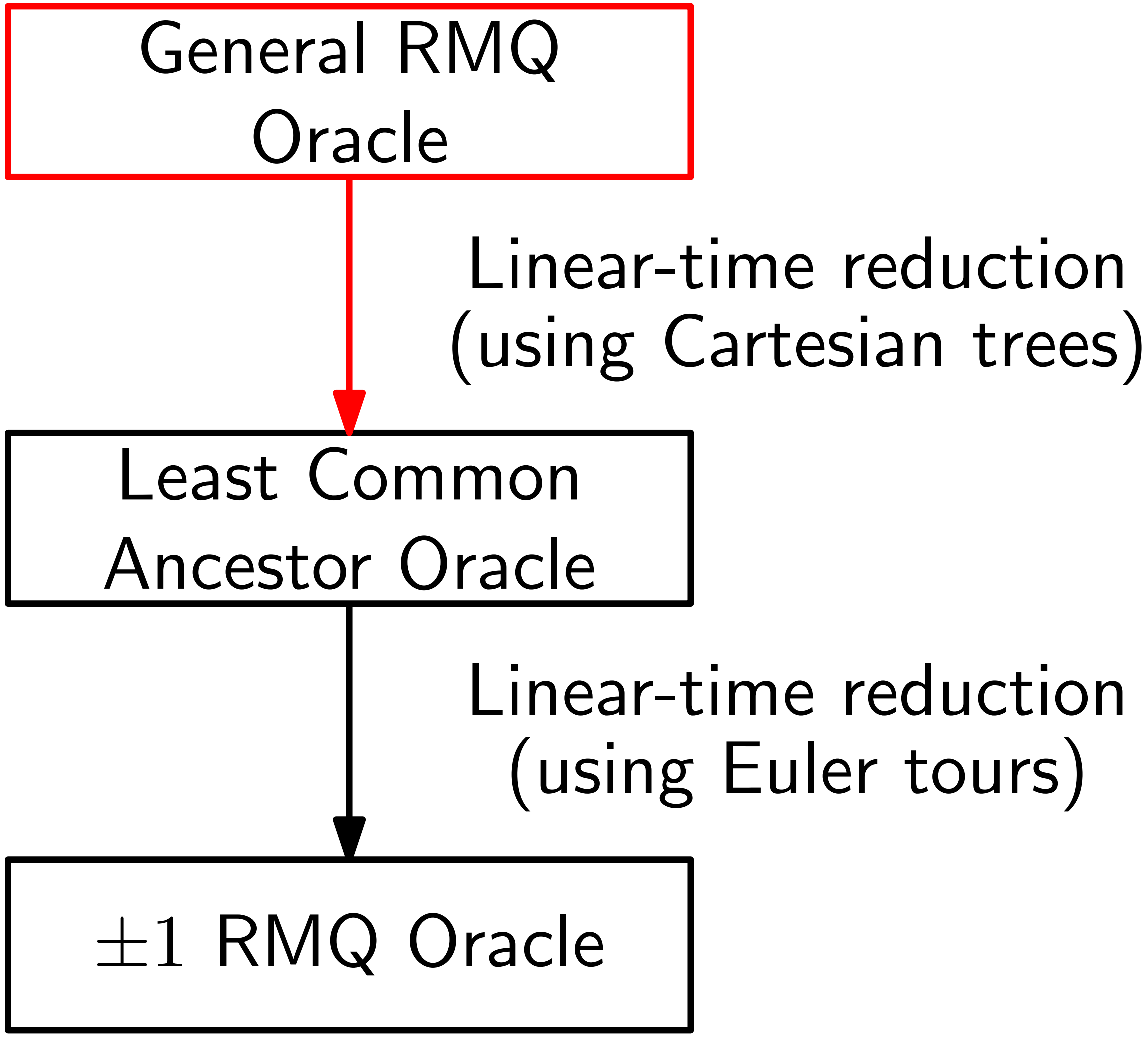
What about the general case?

The General Case



Preprocessing / size $O(n)$.
Query time $O(1)$.

The General Case



Preprocessing / size $O(n)$.
Query time $O(1)$.

Cartesian Trees

	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6

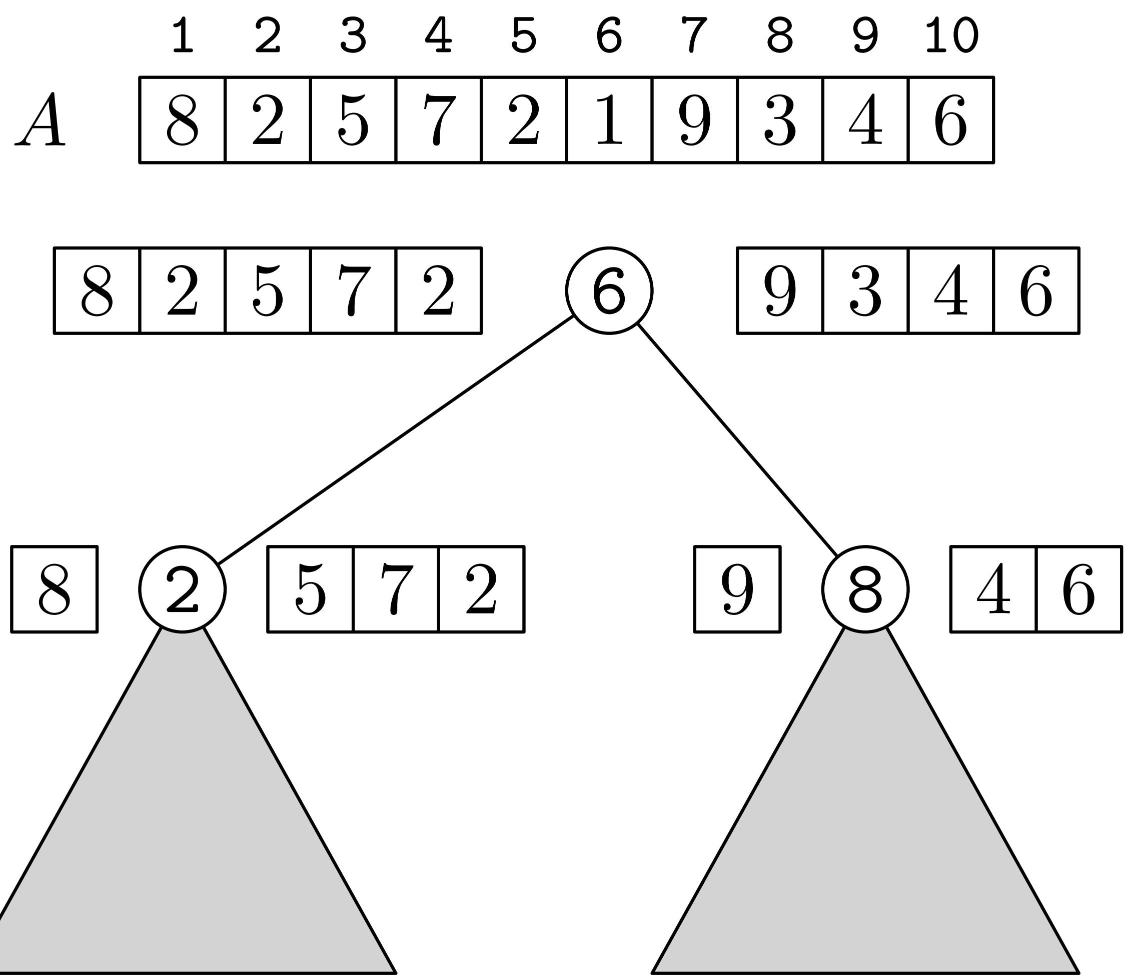
Cartesian Trees

	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6

8	2	5	7	2	(6)	9	3	4	6

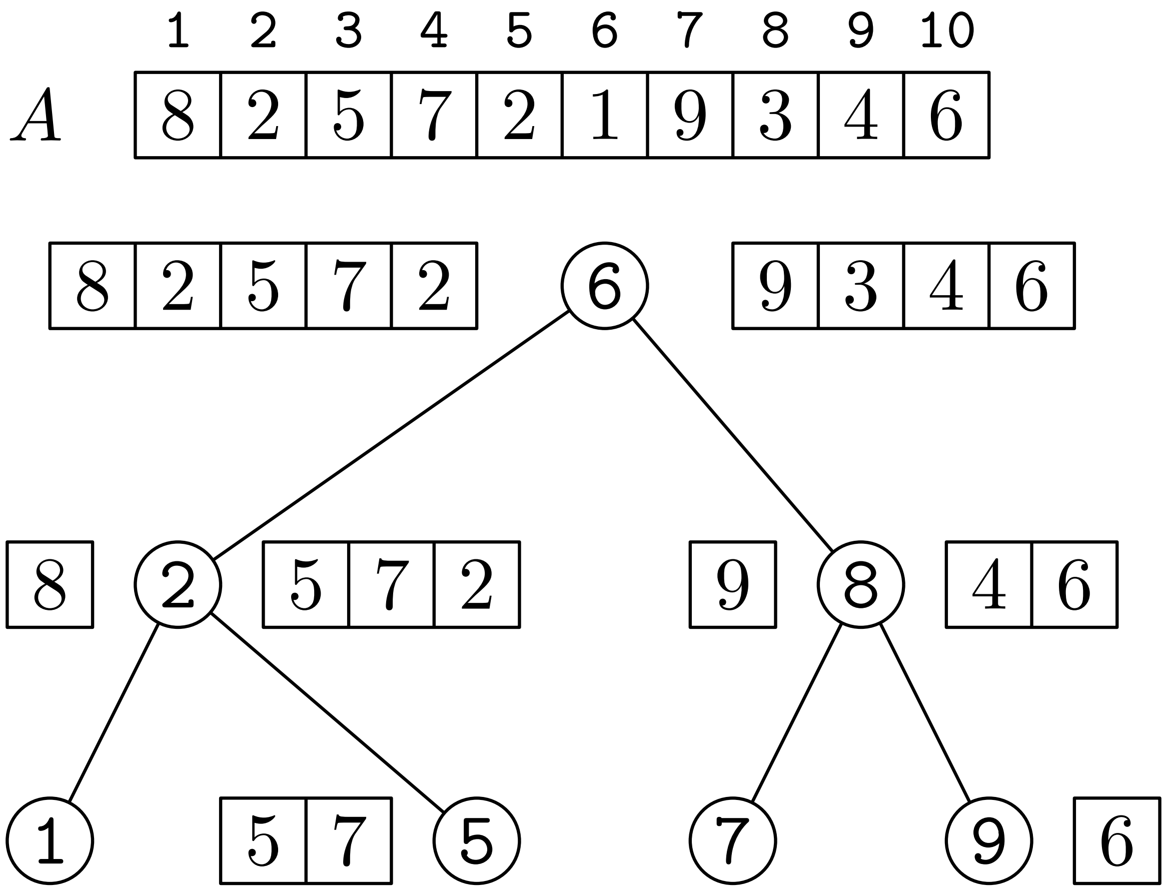
- The root r of the Cartesian tree is the index i of a minimum element a_i of A

Cartesian Trees



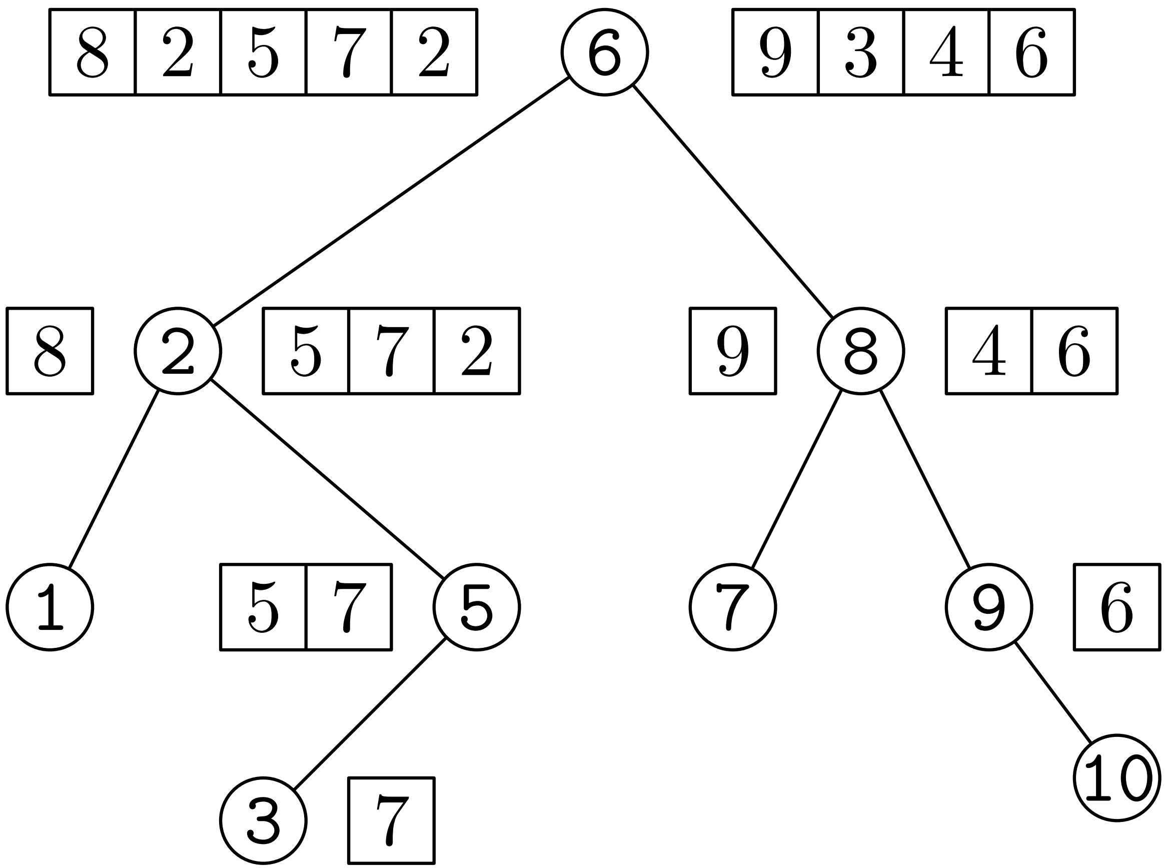
- The root r of the Cartesian tree is the index i of a minimum element a_i of A
- The left and right subtrees r are the Cartesian trees of $A[1 : i-1]$ and $A[i+1 : n]$ (if not empty).

Cartesian Trees



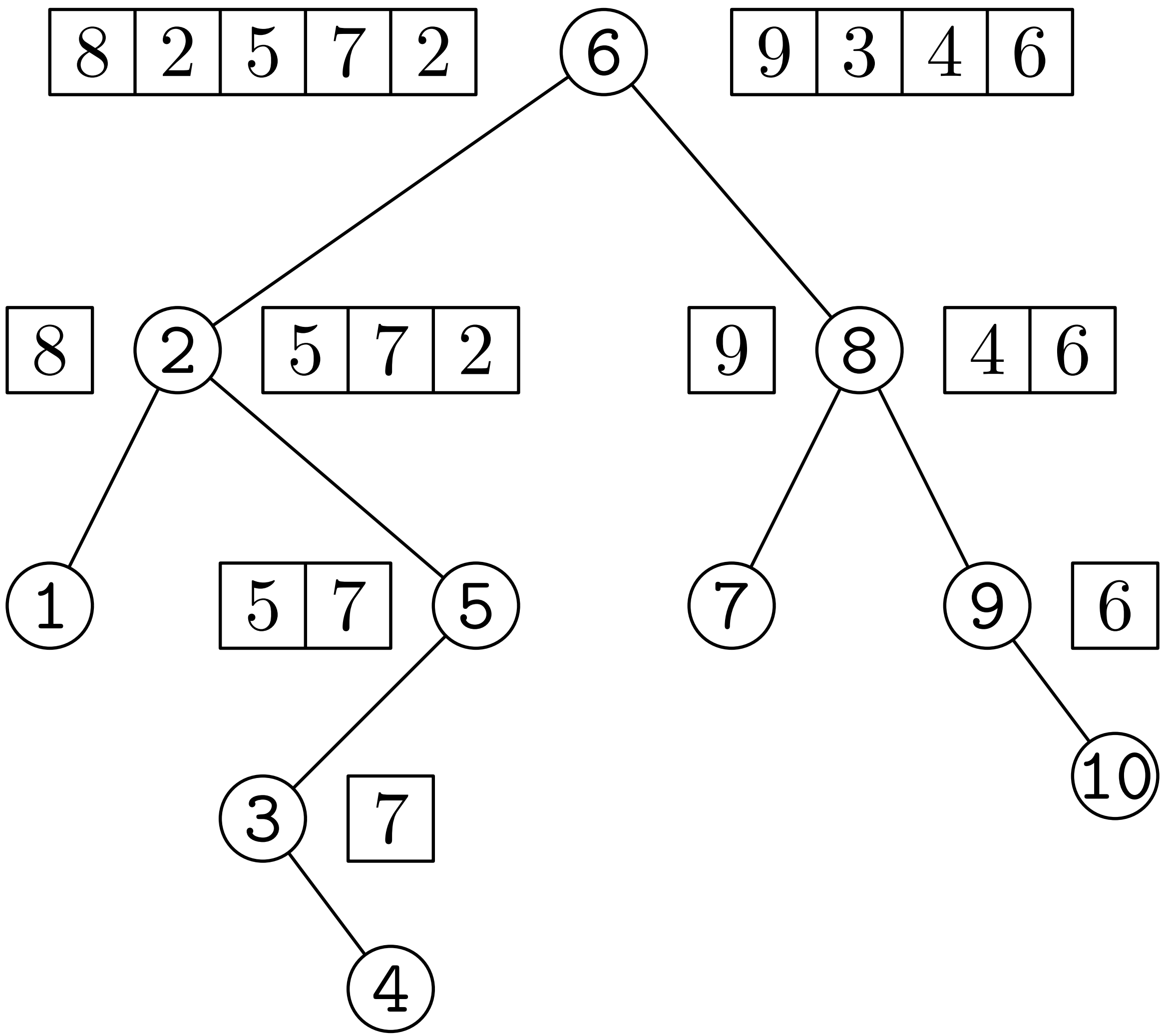
Cartesian Trees

	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



Cartesian Trees

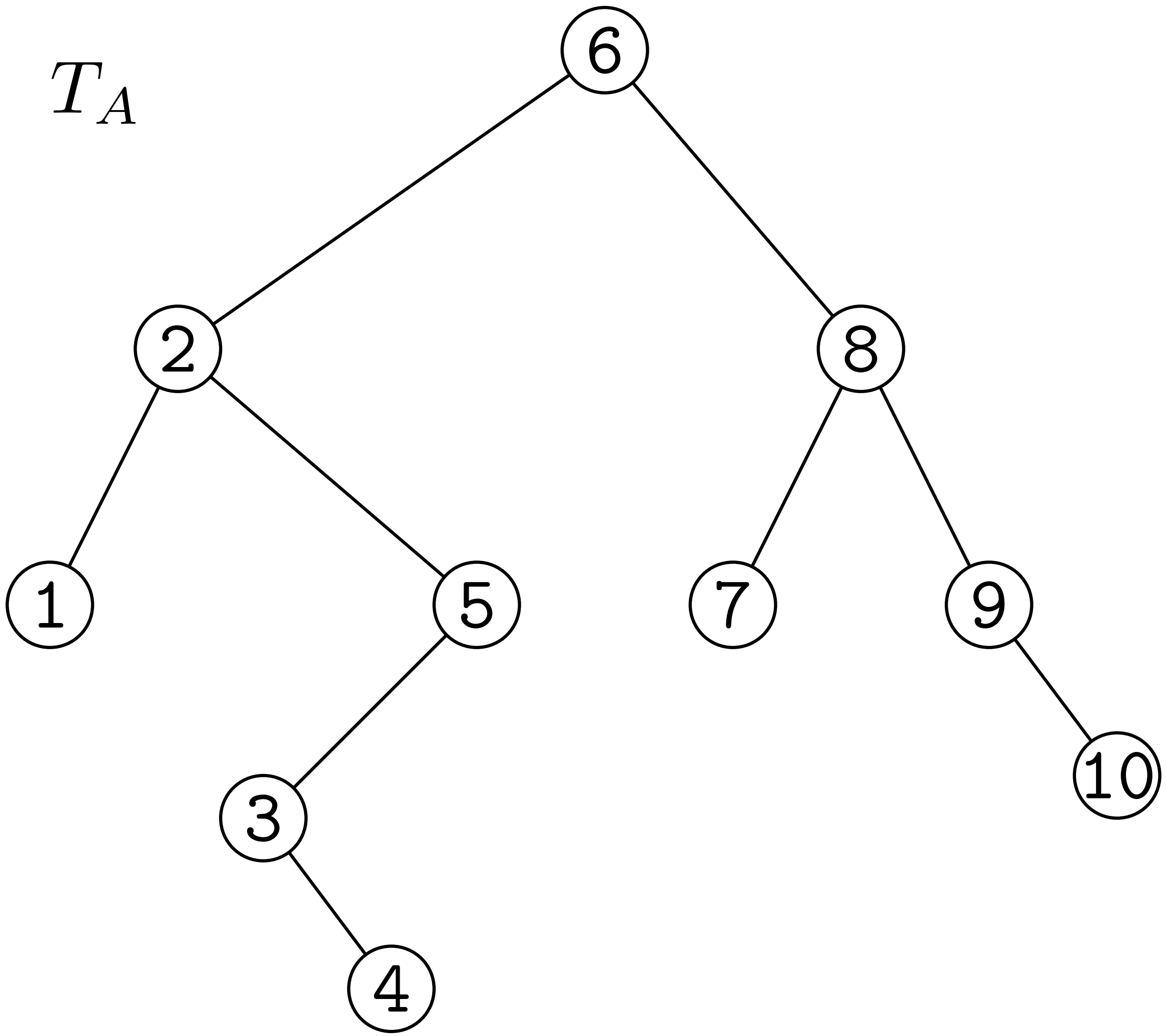
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



Cartesian Trees

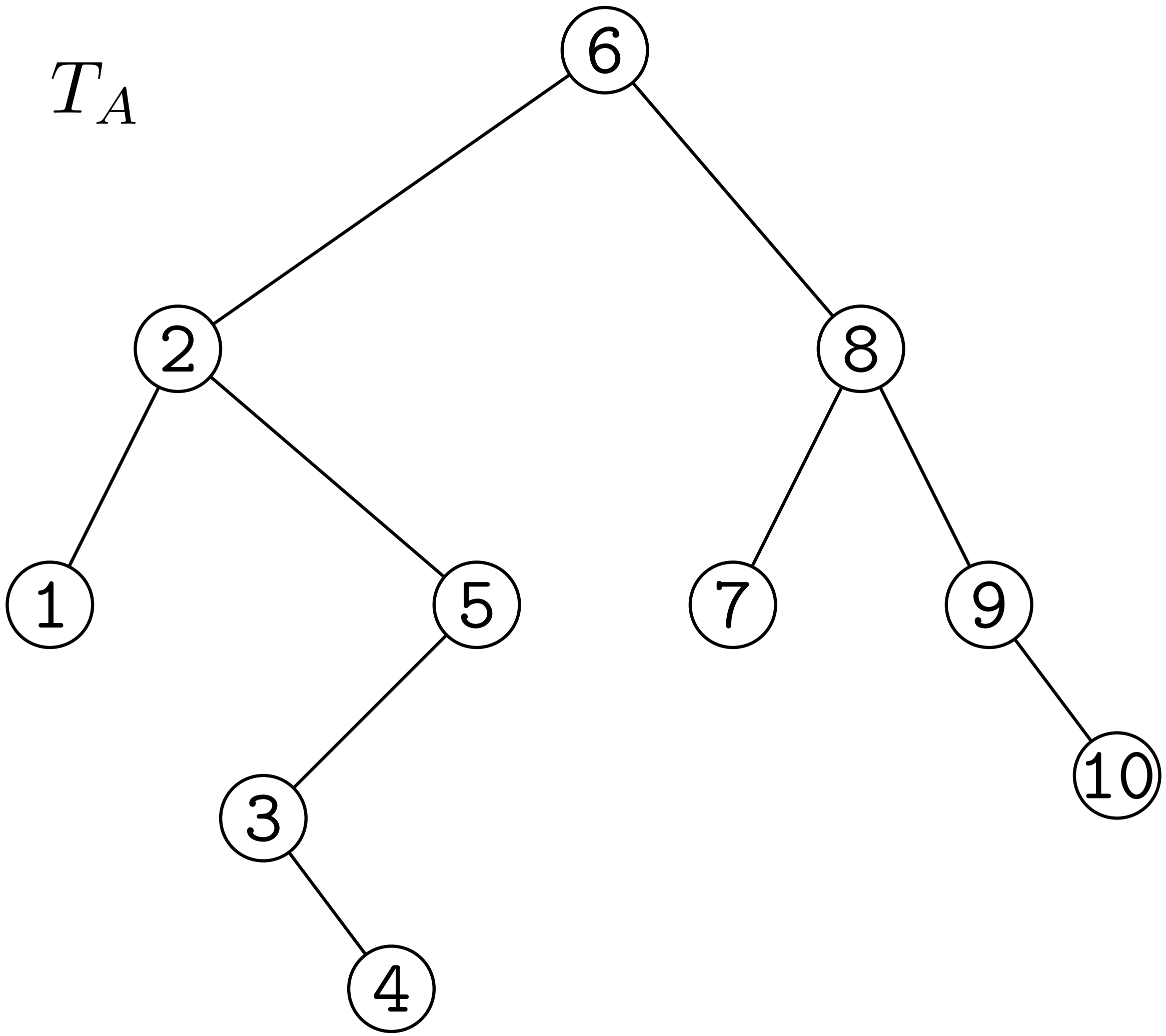
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6

T_A



Cartesian Trees

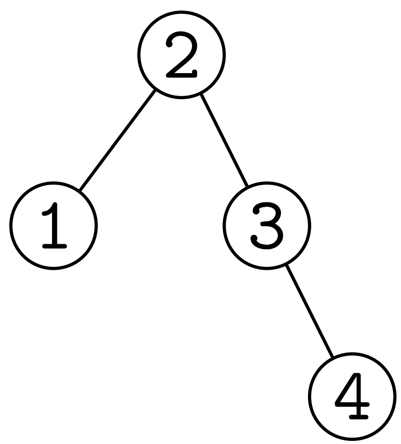
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



Observation: A symmetric visit of T_A visits the nodes in increasing order

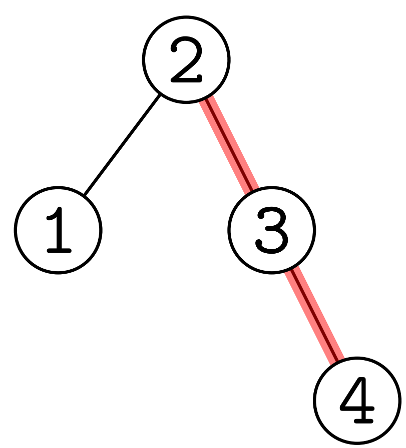
Constructing a Cartesian Tree

	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



Constructing a Cartesian Tree

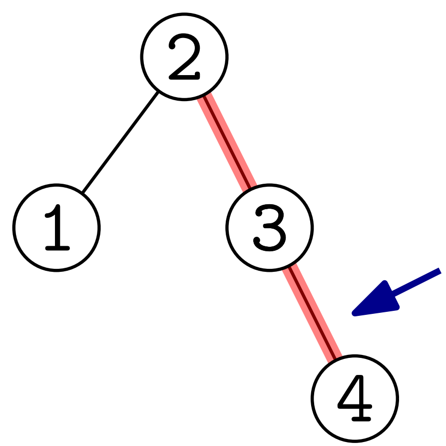
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1:i]$.

Constructing a Cartesian Tree

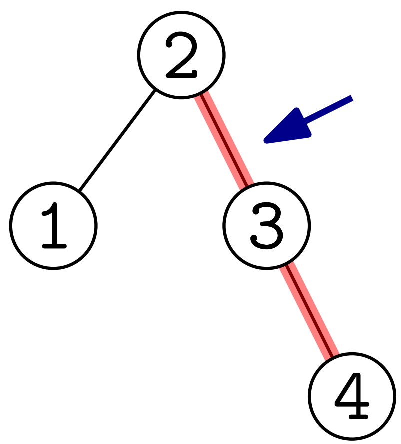
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1:i]$.

Constructing a Cartesian Tree

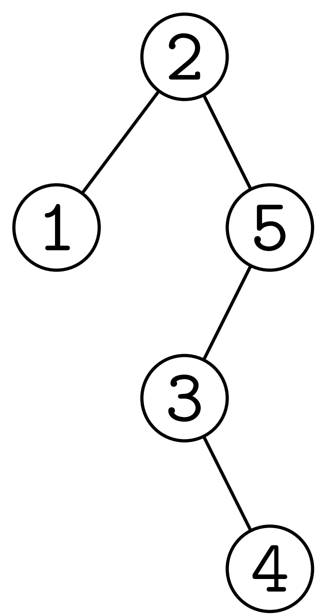
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1 : i]$.

Constructing a Cartesian Tree

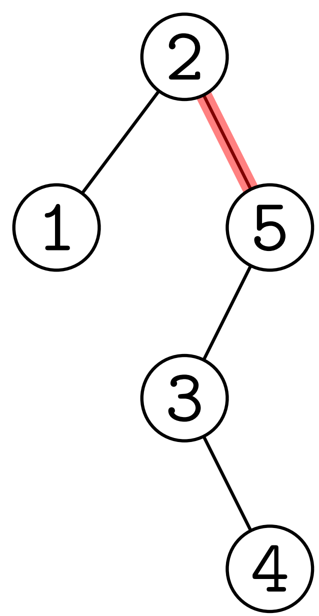
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1:i]$.

Constructing a Cartesian Tree

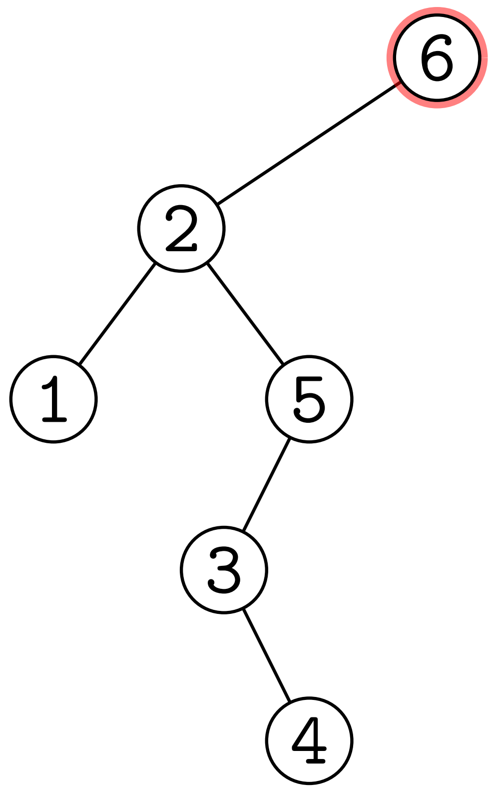
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1:i]$.

Constructing a Cartesian Tree

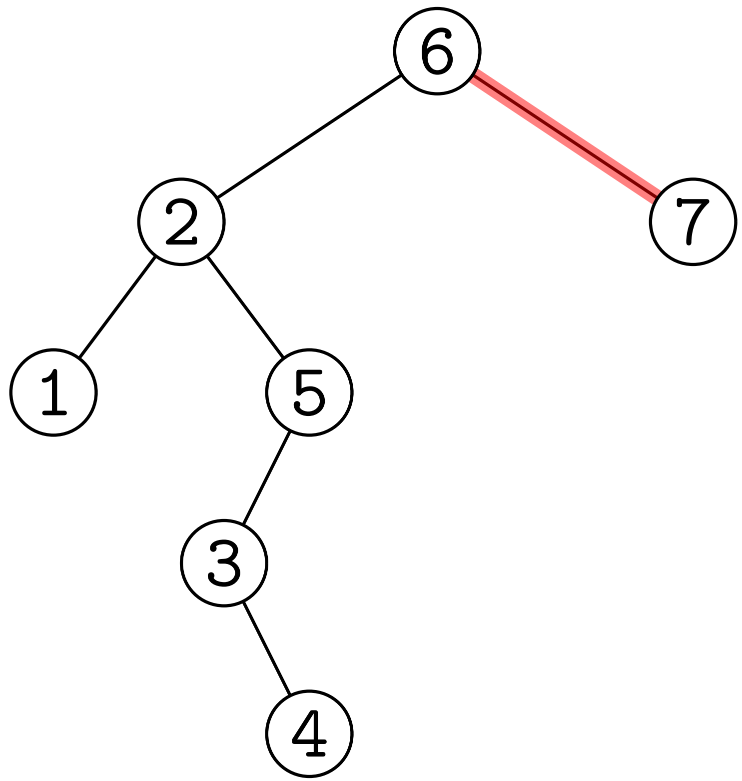
	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1 : i]$.

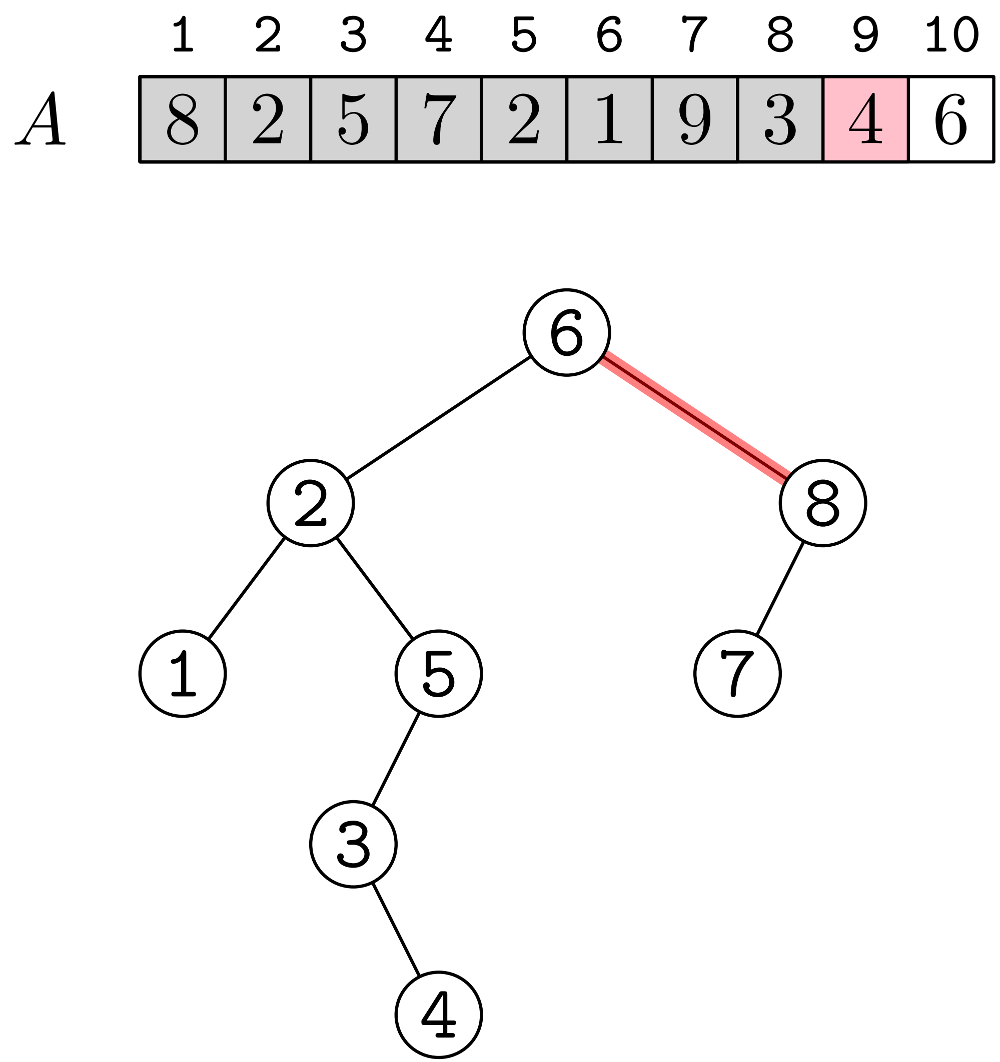
Constructing a Cartesian Tree

	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1:i]$.

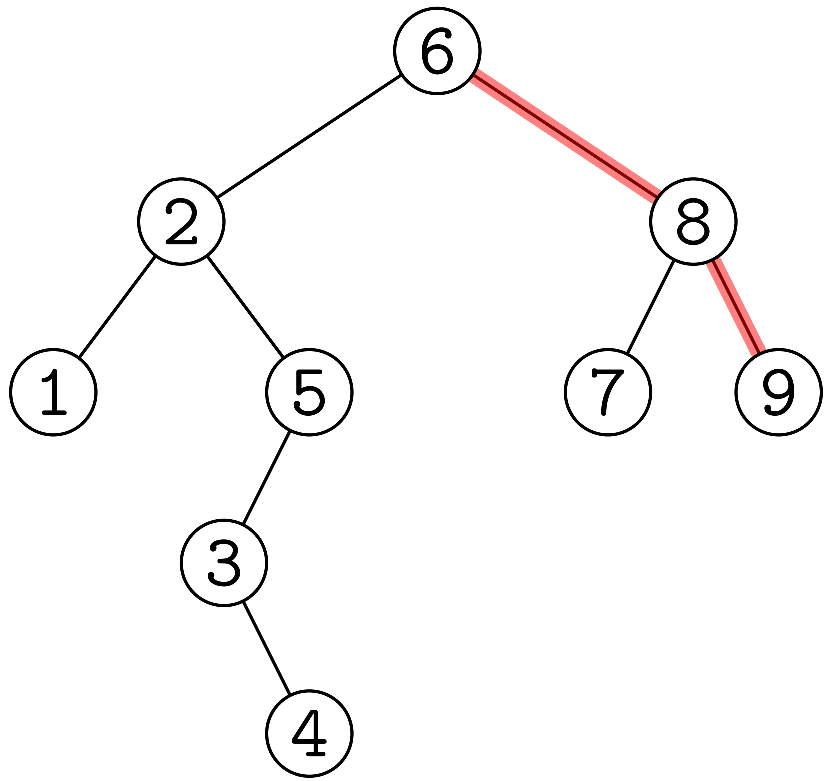
Constructing a Cartesian Tree



The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1 : i]$.

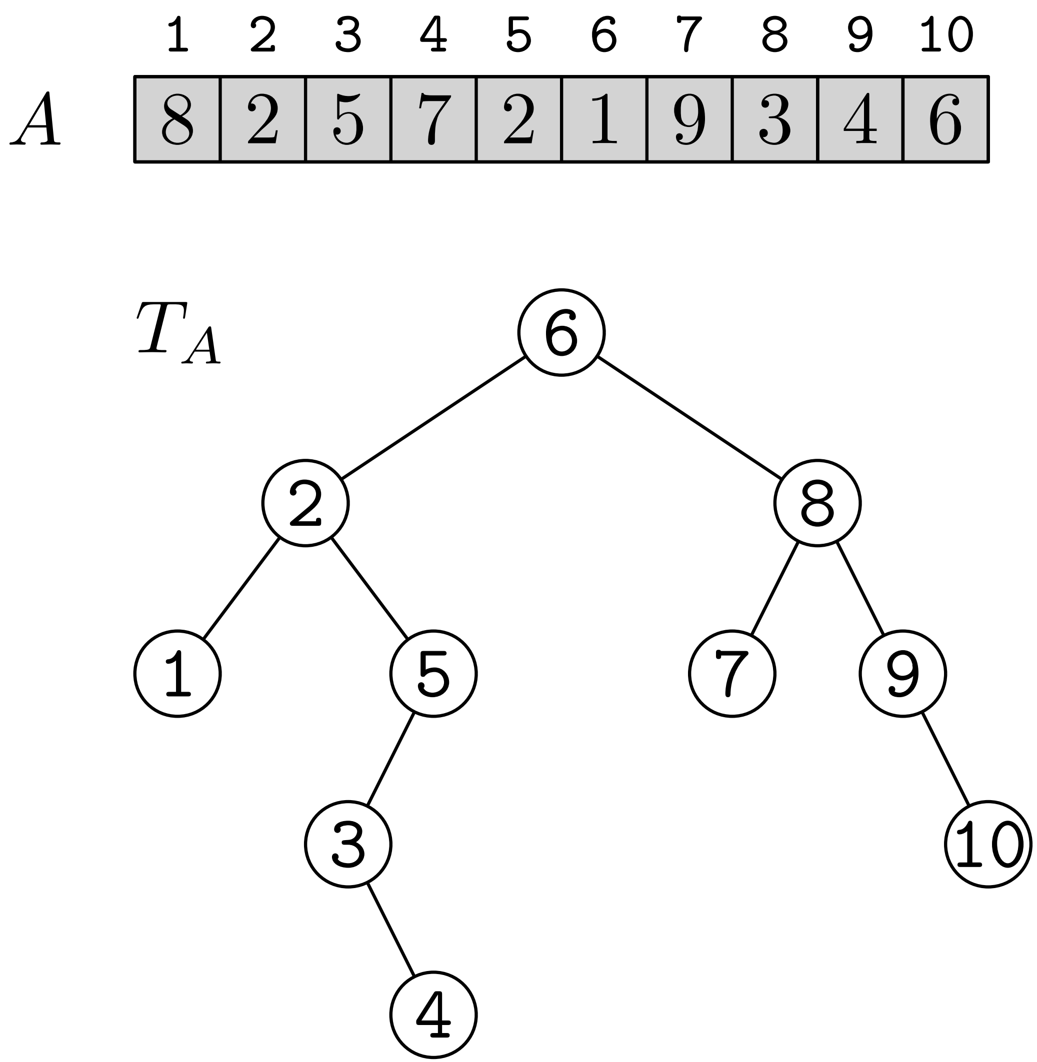
Constructing a Cartesian Tree

	1	2	3	4	5	6	7	8	9	10
A	8	2	5	7	2	1	9	3	4	6



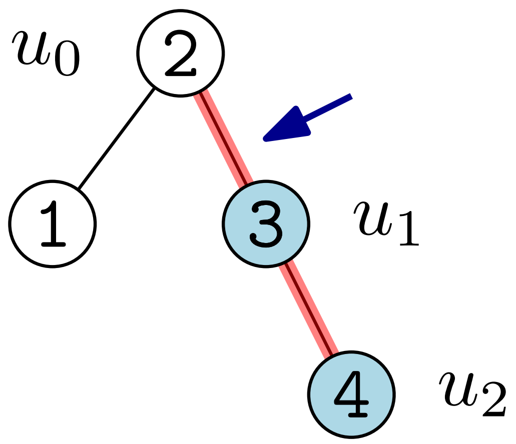
The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1 : i]$.

Constructing a Cartesian Tree



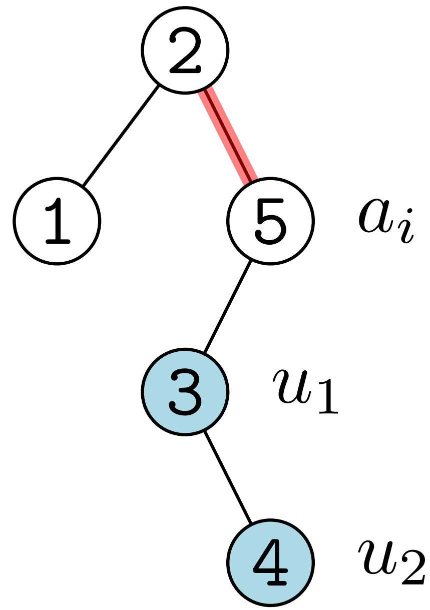
The vertex corresponding to a_{i+1} must belong to the rightmost path of the Cartesian tree of $A[1:i]$.

Constructing a Cartesian Tree



- When a new vertex a_i is inserted, it is compared with $1 + \eta_i$ vertices $u_0, u_1, \ldots, u_{\eta_i}$ on the rightmost path of T.

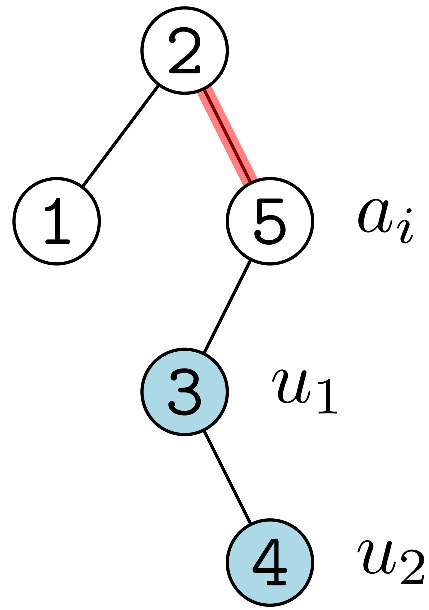
Constructing a Cartesian Tree



- When a new vertex a_i is inserted, it is compared with $1 + \eta_i$ vertices $u_0, u_1, \ldots, u_{\eta_i}$ on the rightmost path of T.

- After a_i is inserted, all vertices $u_1, \ldots, u_{\eta_i}$ will leave the rightmost path of T (and will never join the path again).

Constructing a Cartesian Tree

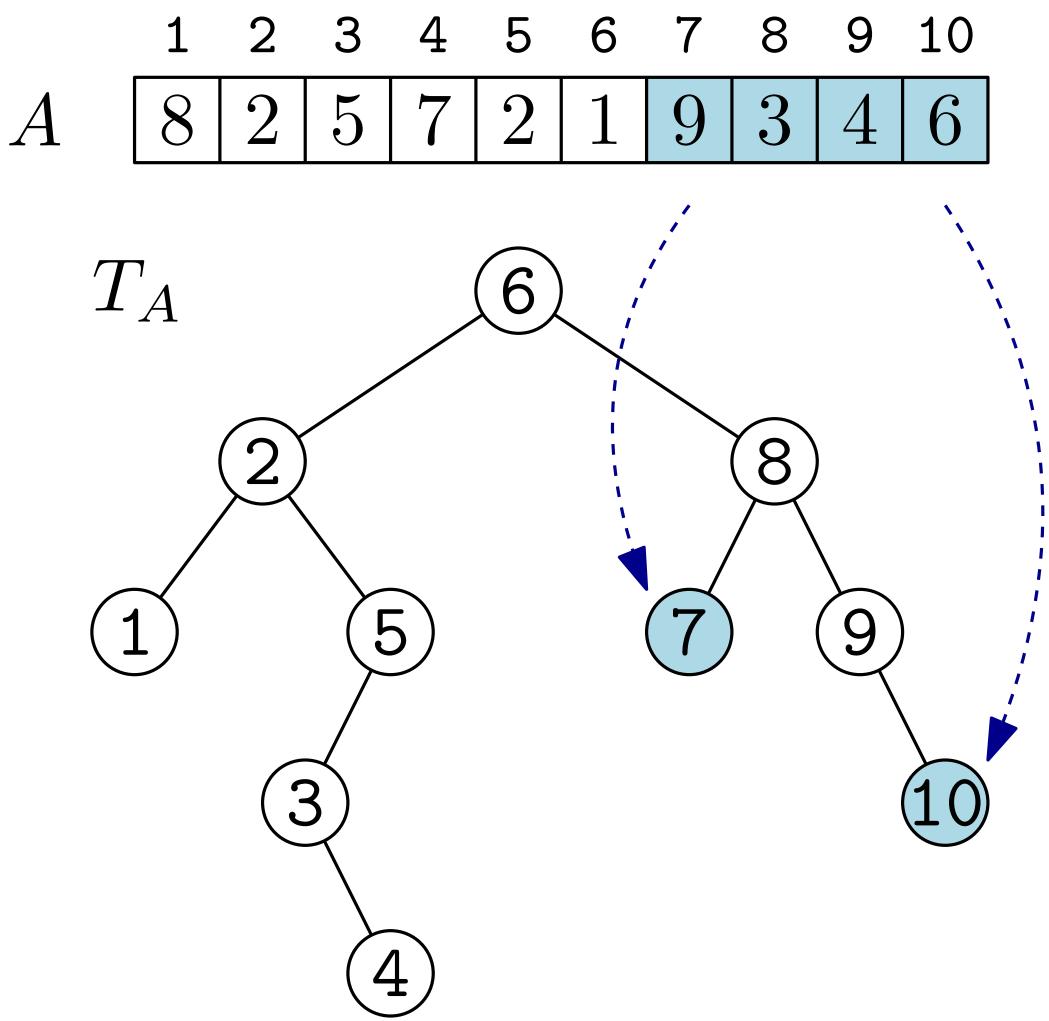


- When a new vertex a_i is inserted, it is compared with $1 + \eta_i$ vertices $u_0, u_1, \ldots, u_{\eta_i}$ on the rightmost path of T.

- After a_i is inserted, all vertices $u_1, \ldots, u_{\eta_i}$ will leave the rightmost path of T (and will never join the path again).

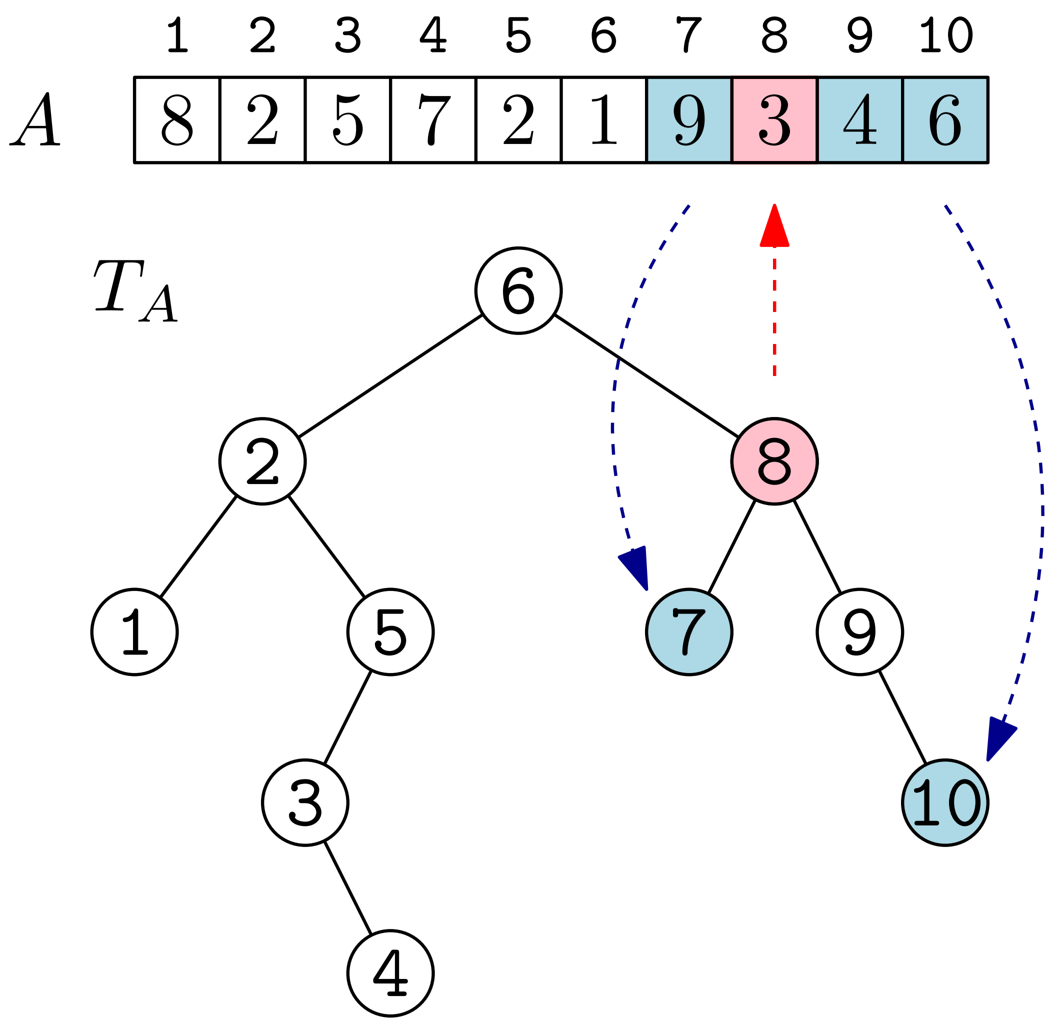
- Total number of comparisons: $\sum_{i=1}^{n}(1 + \eta_i) = n + \sum_{i=1}^{n} \eta_i = n + O(n) = O(n)$.

Cartesian Trees and RMQs



- Let T be the Cartesian tree of A.
- $A[\mathsf{RMQ}(i,j)] = A[\mathsf{LCA}_T(i,j)]$

Cartesian Trees and RMQs

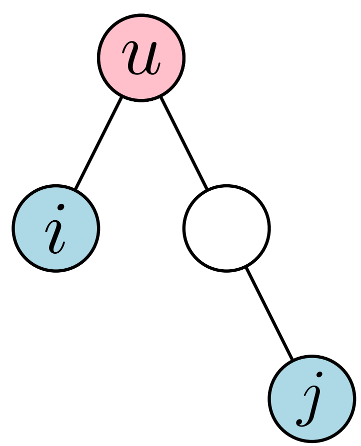


- Let T be the Cartesian tree of A.
- $A[\mathsf{RMQ}(i,j)] = A[\mathsf{LCA}_T(i,j)]$

Cartesian Trees and RMQs

Proof of $A[\text{LCA}_T(i,j)] \geq A[\text{RMQ}(i,j)]$

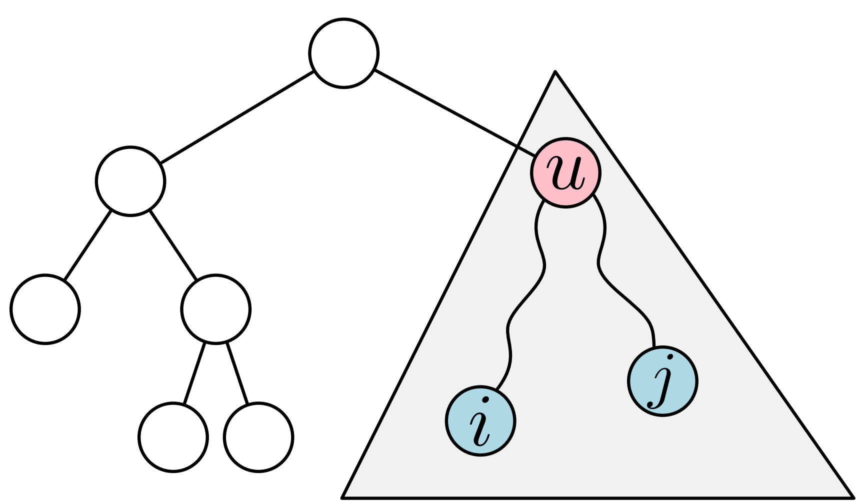
- Let $u = \text{LCA}_T(i,j)$, V_ℓ and V_r be the set vertices in the left and right subtree of u, respectively.
- $i \in V_\ell \cup \{u\}$ and $j \in V_r \cup \{u\}$
- $i \leq u \leq j$
- $A[u] \geq \min A[i:j] = A[\text{RMQ}(i,j)]$



Cartesian Trees and RMQs

Proof of $A[\mathsf{LCA}_T(i,j)] \leq A[\mathsf{RMQ}(i,j)]$

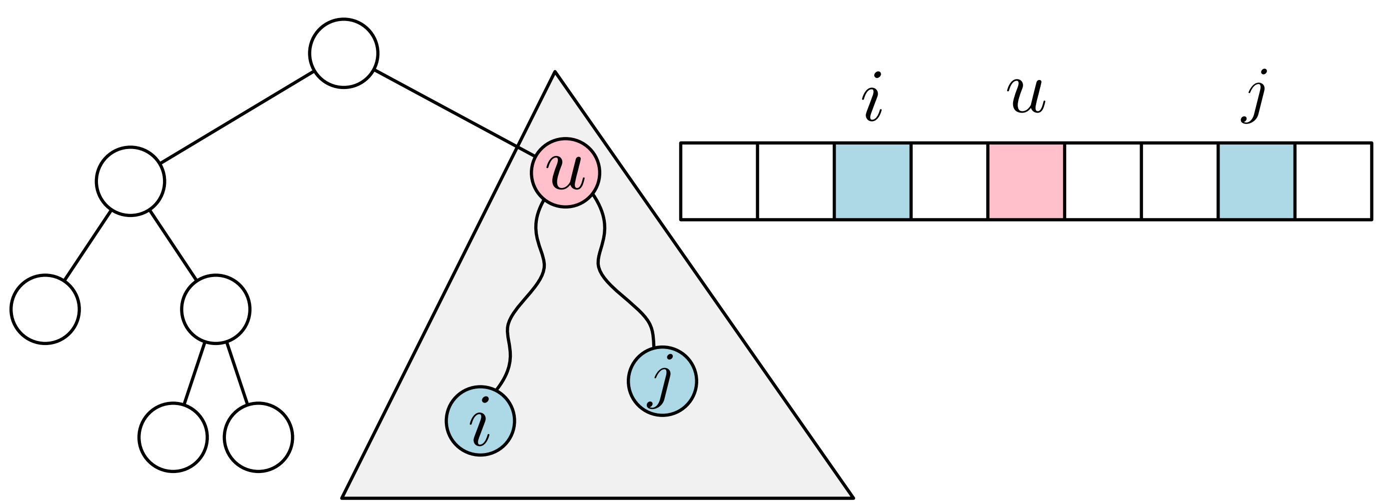
- All vertices k in the subtree T' of T rooted in $\mathsf{LCA}_T(i,j)$ are such that $A[k] \geq A[\mathsf{LCA}_T(i,j)]$



Cartesian Trees and RMQs

Proof of $A[\mathsf{LCA}_T(i,j)] \leq A[\mathsf{RMQ}(i,j)]$

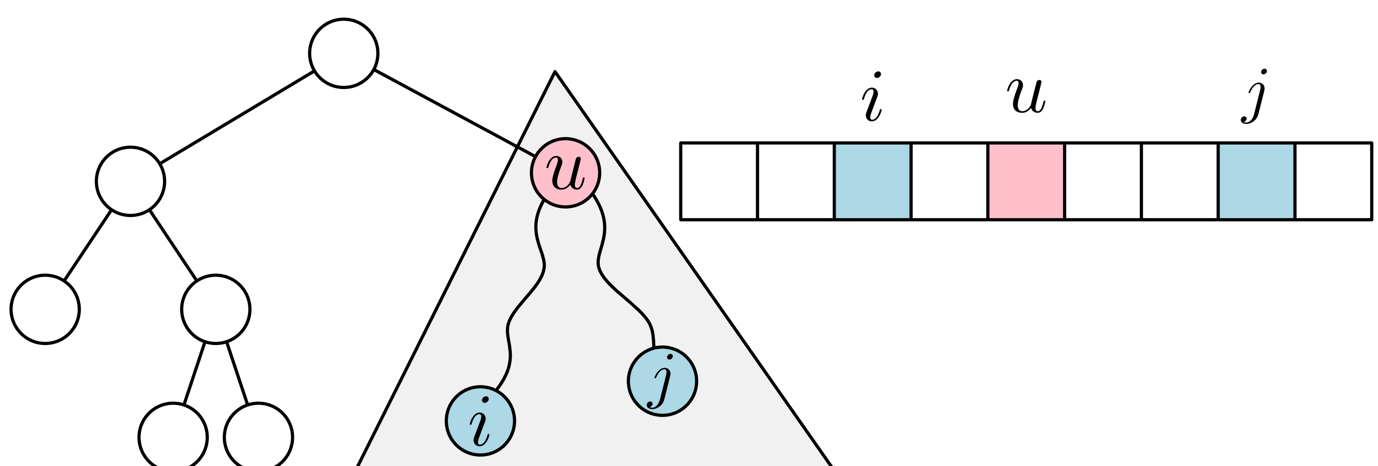
- All vertices k in the subtree T' of T rooted in $\mathsf{LCA}_T(i,j)$ are such that $A[k] \geq A[\mathsf{LCA}_T(i,j)]$
- All subtrees of T correspond to contiguous subarrays of A



Cartesian Trees and RMQs

Proof of $A[\mathsf{LCA}_T(i,j)] \le A[\mathsf{RMQ}(i,j)]$

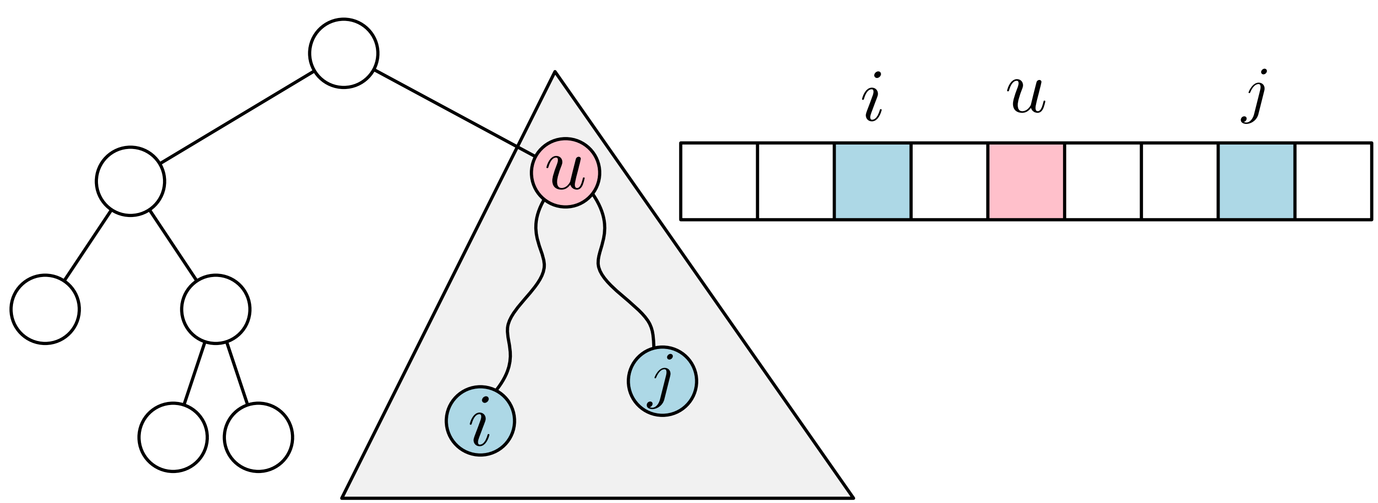
- All vertices k in the subtree T' of T rooted in $\mathsf{LCA}_T(i,j)$ are such that $A[k] \ge A[\mathsf{LCA}_T(i,j)]$
- All subtrees of T correspond to contiguous subarrays of A
- Since $i, j \in T'$, all $k \in \{i, \ldots, j\}$ also belong to T'



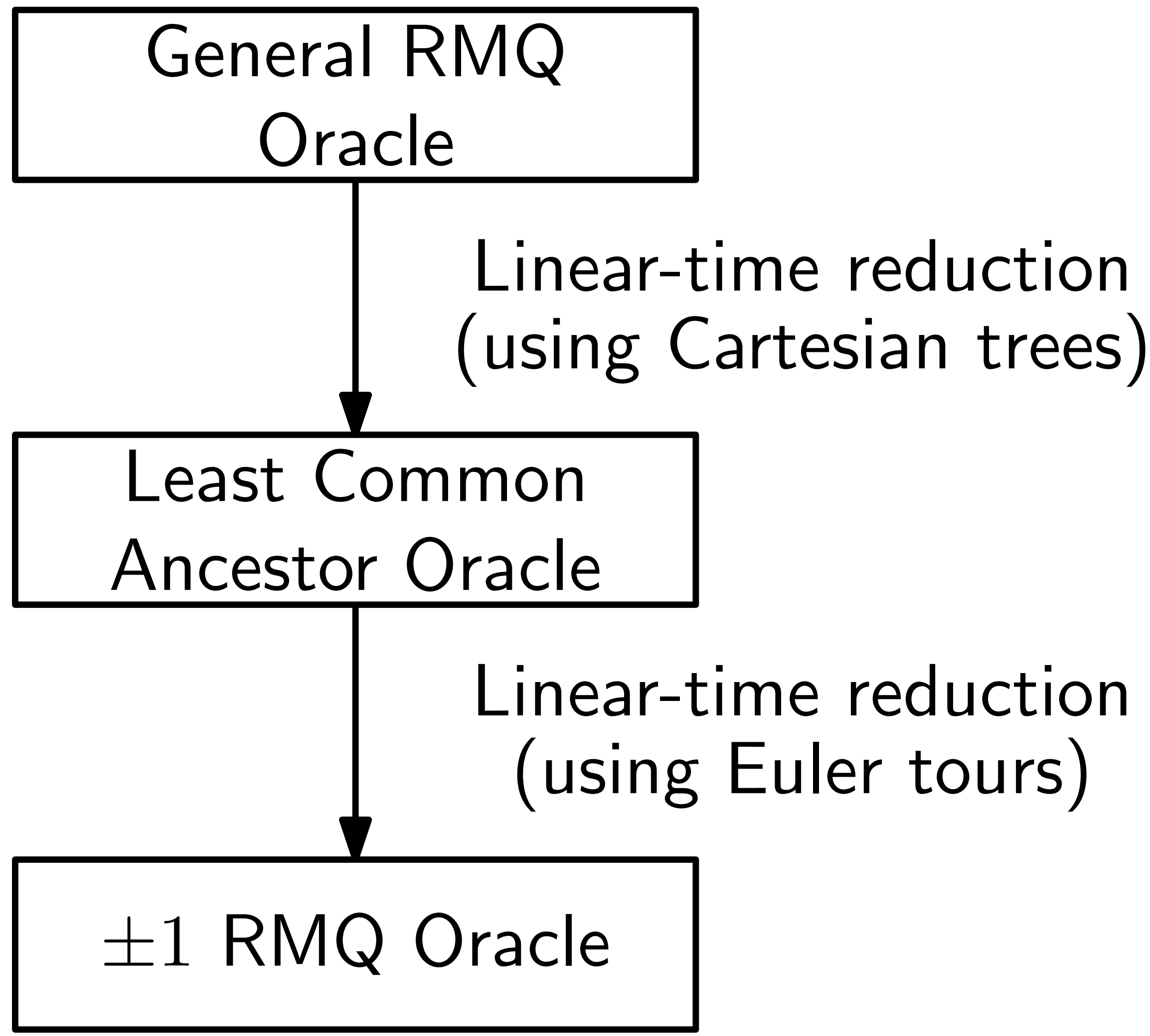
Cartesian Trees and RMQs

Proof of $A[\mathsf{LCA}_T(i,j)] \leq A[\mathsf{RMQ}(i,j)]$

- All vertices k in the subtree T' of T rooted in $\mathsf{LCA}_T(i,j)$ are such that $A[k] \geq A[\mathsf{LCA}_T(i,j)]$
- All subtrees of T correspond to contiguous subarrays of A
- Since $i, j \in T'$, all $k \in \{i, \ldots, j\}$ also belong to T'
- $\mathsf{RMQ}(i,j) \in \{i, \ldots, j\} \Longrightarrow A[\mathsf{RMQ}(i,j)] \geq A[\mathsf{LCA}_T(i,j)]$



The General Case



Preprocessing / size $O(n)$.

Query time $O(1)$.

RMQ Solutions: Recap

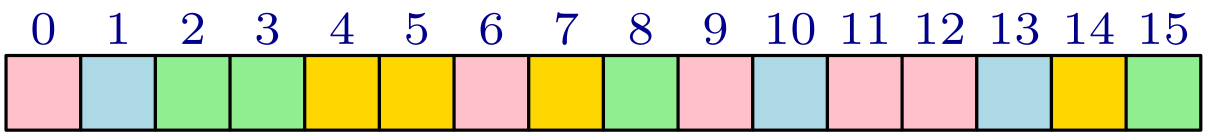
Size	Preprocessing Time	Query Time	Notes
$O(n)$	$O(n)$	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table
$O(n)$	$O(n)$	$O(\log n)$	
$O(n \log \log n)$	$O(n \log \log n)$	$O(1)$	
$O(n)$	$O(n)$	$O(1)$	± 1 RMQ

RMQ Solutions: Recap

Size	Preprocessing Time	Query Time	Notes
$O(n)$	$O(n)$	$O(n)$	
$O(n^2)$	$O(n^3)$	$O(1)$	
$O(n^2)$	$O(n^2)$	$O(1)$	
$O(n \log n)$	$O(n \log n)$	$O(1)$	Sparse Table
$O(n)$	$O(n)$	$O(\log n)$	
$O(n \log \log n)$	$O(n \log \log n)$	$O(1)$	
$O(n)$	$O(n)$	$O(1)$	± 1 RMQ
$O(n)$	$O(n)$	$O(1)$	General case

Finding Distinct Items in a Range

Input: An array A of not necessarily distinct items (colors).

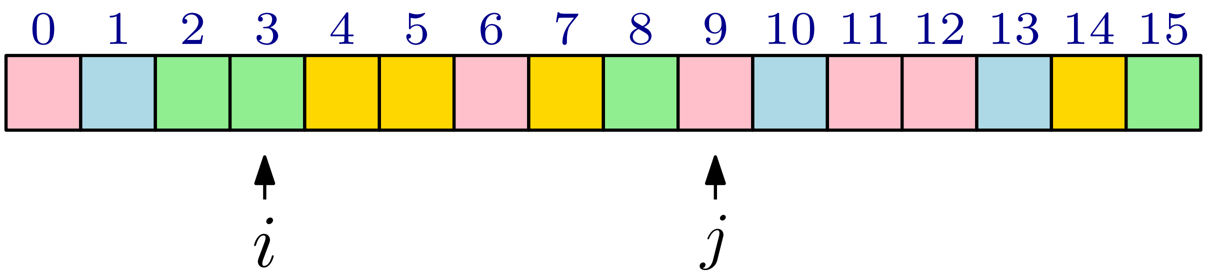


Goal: Preprocess A to answer queries of the following form:

Given two indices i, j, find the distinct items (colors) in $A[i, j]$ and, for each of them, return the index of its first occurrence in $A[i, j]$.

Finding Distinct Items in a Range

Input: An array A of not necessarily distinct items (colors).

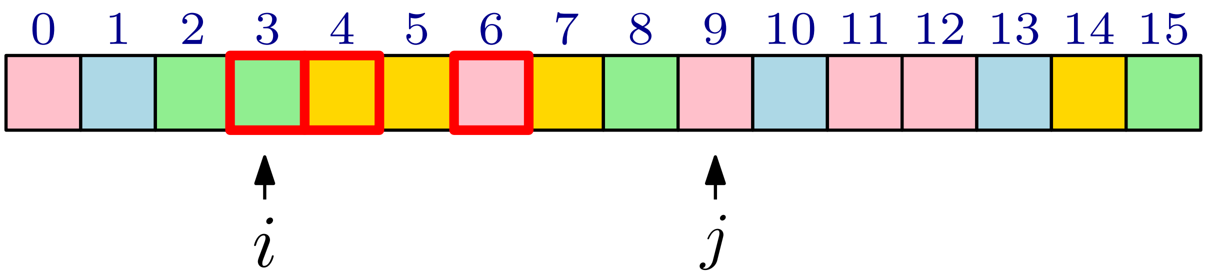


Goal: Preprocess A to answer queries of the following form:

Given two indices i, j, find the distinct items (colors) in $A[i, j]$ and, for each of them, return the index of its first occurrence in $A[i, j]$.

Finding Distinct Items in a Range

Input: An array A of not necessarily distinct items (colors).

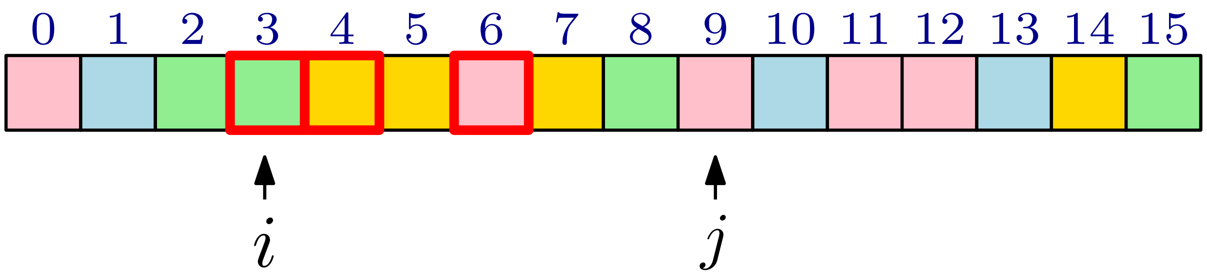


Goal: Preprocess A to answer queries of the following form:

Given two indices i, j, find the distinct items (colors) in $A[i, j]$ and, for each of them, return the index of its first occurrence in $A[i, j]$.

Finding Distinct Items in a Range

Input: An array A of not necessarily distinct items (colors).



Goal: Preprocess A to answer queries of the following form:

Given two indices i, j, find the distinct items (colors) in $A[i, j]$ and, for each of them, return the index of its first occurrence in $A[i, j]$.

Target time complexity: $O(\#\text{returned items})$

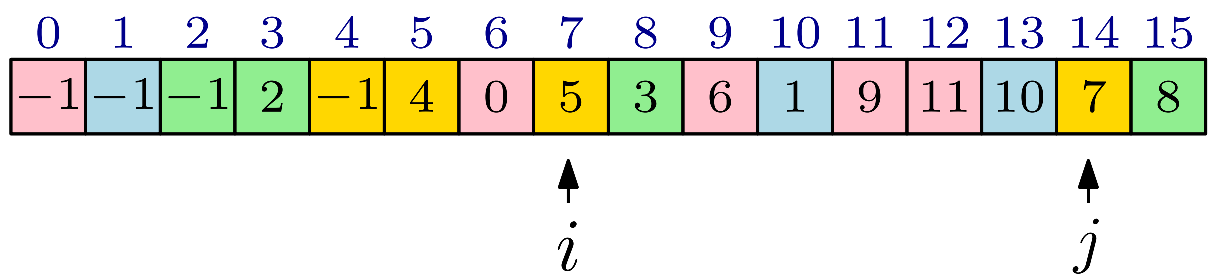
Finding Distinct Items in a Range

Hint 1: Label each $A[h]$ with the largest index $\ell_h < h$ such that $A[\ell_h] = A[h]$ (or -1 if no such index exists).

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

Finding Distinct Items in a Range

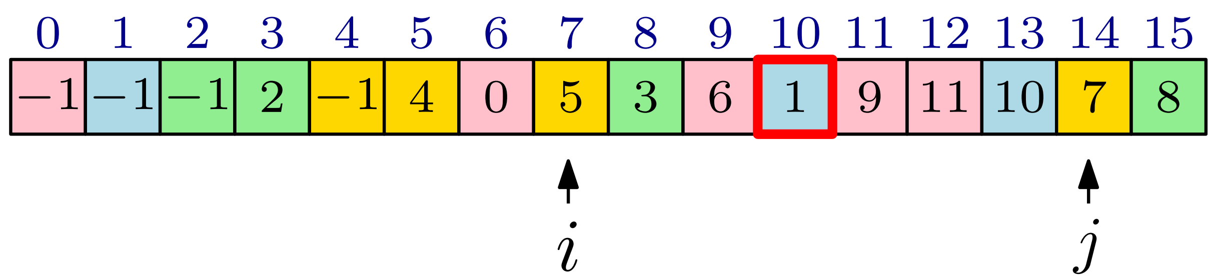
Hint 1: Label each $A[h]$ with the largest index $\ell_h < h$ such that $A[\ell_h] = A[h]$ (or -1 if no such index exists).



Hint 2: For $i \leq h \leq j$, $A[h]$ is the first occurence of an item in $A[i:j]$ iff $\ell_h < i$.

Finding Distinct Items in a Range

Hint 1: Label each $A[h]$ with the largest index $\ell_h < h$ such that $A[\ell_h] = A[h]$ (or -1 if no such index exists).



Hint 2: For $i \leq h \leq j$, $A[h]$ is the first occurence of an item in $A[i:j]$ iff $\ell_h < i$.

Hint 3: The index h such that $i \leq h \leq j$ that minimizes ℓ_h is the first occurrence of some item. How should $A[i : h-1]$ and $A[h+1 : j]$ be handled?