Range Minimum and Lowest Common Ancestor Queries

Slides by Solon P. Pissis

November 15, 2019

Range Minimum Queries problem

Range Minimum Queries problem

Definition

Given an array $A[1 \ldots n]$, preprocess A so that a minimum of any fragment $A[i \ldots j]$ can be computed efficiently:

$$\mathrm{RMQ}_A(i, j) = \text{arg min } A[k]$$

with $1 \leq i \leq k \leq j \leq n$.

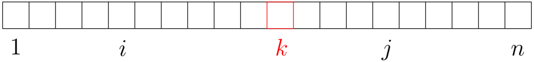
Range Minimum Queries problem

Definition

Given an array $A[1 \ldots n]$, preprocess A so that a minimum of any fragment $A[i \ldots j]$ can be computed efficiently:

$$\mathrm{RMQ}_A(i, j) = \arg\min A[k]$$

with $1 \leq i \leq k \leq j \leq n$.



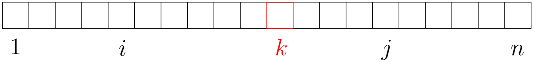
Range Minimum Queries problem

Definition

Given an array $A[1 \dots n]$, preprocess A so that a minimum of any fragment $A[i \dots j]$ can be computed efficiently:

$$\mathrm{RMQ}_A(i, j) = \arg\min A[k]$$

with $1 \leq i \leq k \leq j \leq n$.



Answering any query in $O(1)$ time is trivial if we allow $O(n^2)$ time and space preprocessing.

$O(n \log n)$ time and space and $O(1)$-time queries

$O(n \log n)$ time and space and $O(1)$-time queries

Let us show the following lemma.

$O(n \log n)$ time and space and $O(1)$-time queries

Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(1)$ time after $O(n \log n)$ time and space preprocessing.

$O(n \log n)$ time and space and $O(1)$-time queries

Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(1)$ time after $O(n \log n)$ time and space preprocessing.

We apply the well-known *doubling* technique for **preprocessing**.

$O(n \log n)$ time and space and $O(1)$-time queries

Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(1)$ time after $O(n \log n)$ time and space preprocessing.

We apply the well-known *doubling* technique for **preprocessing**.

For all $k = 0, 1, \ldots, \log n$ construct:

$$B_k[i] = \min\{A[i], A[i+1], \ldots, A[i+2^k-1]\}.$$

$O(n \log n)$ time and space and $O(1)$-time queries

Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(1)$ time after $O(n \log n)$ time and space preprocessing.

We apply the well-known *doubling* technique for **preprocessing**.

For all $k = 0, 1, \ldots, \log n$ construct:

$$B_k[i] = \min\{A[i], A[i+1], \ldots, A[i + 2^k - 1]\}.$$

How?

$O(n \log n)$ time and space and $O(1)$-time queries

Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(1)$ time after $O(n \log n)$ time and space preprocessing.

We apply the well-known *doubling* technique for **preprocessing**.

For all $k = 0, 1, \ldots, \log n$ construct:

$$B_k[i] = \min\{A[i], A[i+1], \ldots, A[i+2^k-1]\}.$$

How? $B_0[i] = A[i]$ and $B_{k+1}[i] = \min\{B_k[i], B_k[i+2^k]\}$, for all i.

$O(n \log n)$ time and space and $O(1)$-time queries

And now the **querying** part.

$O(n \log n)$ time and space and $O(1)$-time queries

And now the **querying** part.

Two cases:

$O(n \log n)$ time and space and $O(1)$-time queries

And now the **querying** part.

Two cases:

- Fragment of length 2^k: Trivial, use B;

$O(n \log n)$ time and space and $O(1)$-time queries

And now the **querying** part.

Two cases:

- Fragment of length 2^k: Trivial, use B;
- Otherwise, we can cover any range with two power-of-2 ranges.

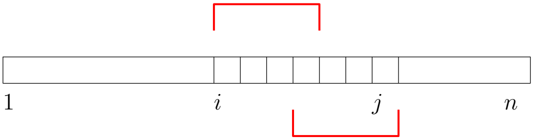
$O(n \log n)$ time and space and $O(1)$-time queries

And now the **querying** part.

Two cases:

- Fragment of length 2^k: Trivial, use B;
- Otherwise, we can cover any range with two power-of-2 ranges.

Compute $k = \lfloor \log(j - i + 1) \rfloor$.



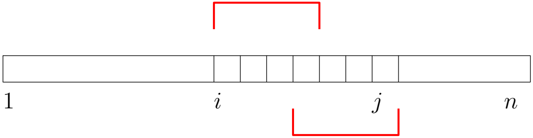
$O(n \log n)$ time and space and $O(1)$-time queries

And now the **querying** part.

Two cases:

- Fragment of length 2^k: Trivial, use B;
- Otherwise, we can cover any range with two power-of-2 ranges.

Compute $k = \lfloor \log(j - i + 1) \rfloor$.



Return $\min\{B_k[i], B_k[j - 2^k + 1]\}$ in $O(1)$ time!

$O(n)$ time and space and $O(\log n)$-time queries

$O(n)$ time and space and $O(\log n)$-time queries

Let us show the following lemma.

$O(n)$ time and space and $O(\log n)$-time queries

Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(\log n)$ time after $O(n)$ time and space preprocessing.

$O(n)$ time and space and $O(\log n)$-time queries

Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(\log n)$ time after $O(n)$ time and space preprocessing.

(Segment tree.)

$O(n)$ time and space and $O(\log n)$-time queries

Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(\log n)$ time after $O(n)$ time and space preprocessing.

(Segment tree.)

We apply the well-known *micro-macro decomposition* technique.

$O(n)$ time and space and $O(\log n)$-time queries

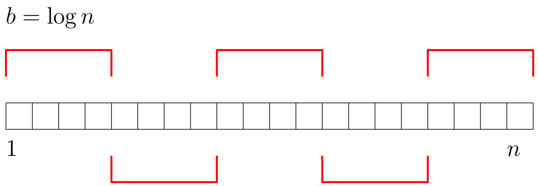
Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(\log n)$ time after $O(n)$ time and space preprocessing.

(Segment tree.)

We apply the well-known *micro-macro decomposition* technique.



$O(n)$ time and space and $O(\log n)$-time queries

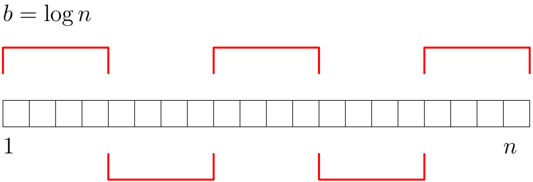
Let us show the following lemma.

Lemma

The RMQ problem can be solved in $O(\log n)$ time after $O(n)$ time and space preprocessing.

(Segment tree.)

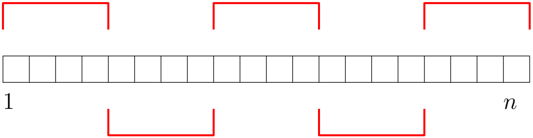
We apply the well-known *micro-macro decomposition* technique.



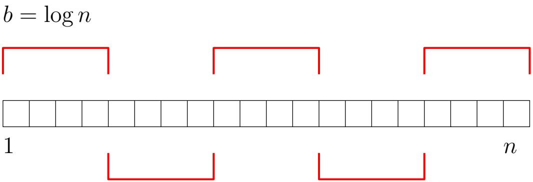
Decompose array A into blocks of length $b = \log n$.

$O(n)$ time and space and $O(\log n)$-time queries

$b = \log n$



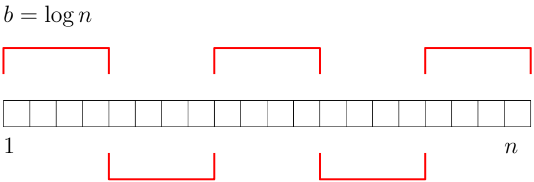
$O(n)$ time and space and $O(\log n)$-time queries



- Construct a new array A':

$$A'[i] = \min\{A[i \cdot b + 1], A[i \cdot b + 2], \ldots, A[(i+1) \cdot b]\}.$$

$O(n)$ time and space and $O(\log n)$-time queries

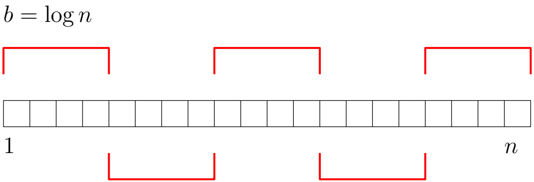


- Construct a new array A':

$$A'[i] = \min\{A[i \cdot b + 1], A[i \cdot b + 2], \ldots, A[(i + 1) \cdot b]\}.$$

- Build the previously described structure for $A'[1 \ldots \frac{n}{\log n}]$.

$O(n)$ time and space and $O(\log n)$-time queries



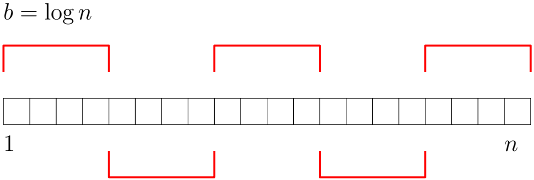
- Construct a new array A':

$$A'[i] = \min\{A[i \cdot b + 1], A[i \cdot b + 2], \ldots, A[(i+1) \cdot b]\}.$$

- Build the previously described structure for $A'[1 \ldots \frac{n}{\log n}]$.

How much time and space do we need?

$O(n)$ time and space and $O(\log n)$-time queries



- Construct a new array A':

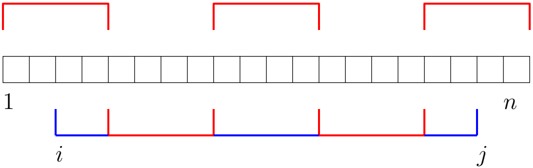
$$A'[i] = \min\{A[i \cdot b + 1], A[i \cdot b + 2], \ldots, A[(i+1) \cdot b]\}.$$

- Build the previously described structure for $A'[1 \ldots \frac{n}{\log n}]$.

How much time and space do we need? $O(\frac{n}{\log n} \log(\frac{n}{\log n})) = O(n)$.

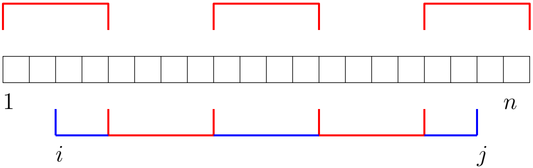
$O(n)$ time and space and $O(\log n)$-time queries

Let i, j be a query (Notice: endpoints are inside blocks).



$O(n)$ time and space and $O(\log n)$-time queries

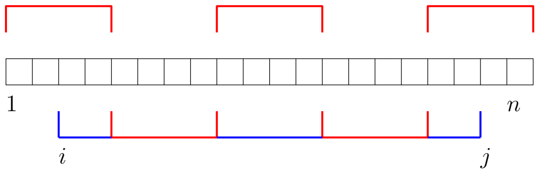
Let i, j be a query (Notice: endpoints are inside blocks).



- We do not have the min in the prefix and suffix of every block.

$O(n)$ time and space and $O(\log n)$-time queries

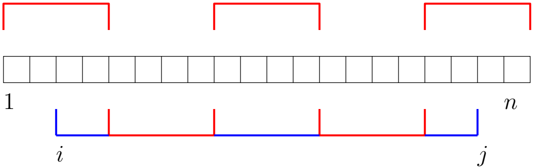
Let i, j be a query (Notice: endpoints are inside blocks).



- We do not have the min in the prefix and suffix of every block.
- For each block, precompute it: $O(n)$ time and space!

$O(n)$ time and space and $O(\log n)$-time queries

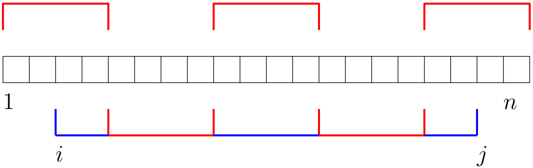
Let i, j be a query (Notice: endpoints are inside blocks).



- We do not have the min in the prefix and suffix of every block.
- For each block, precompute it: $O(n)$ time and space!
- For the rest: use the structure we have built for A'.

$O(n)$ time and space and $O(\log n)$-time queries

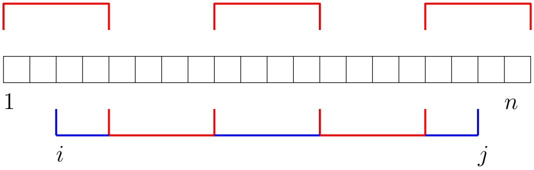
Let i, j be a query (Notice: endpoints are inside blocks).



- We do not have the min in the prefix and suffix of every block.
- For each block, precompute it: $O(n)$ time and space!
- For the rest: use the structure we have built for A'.
- Now any such query takes $O(1)$ time.

$O(n)$ time and space and $O(\log n)$-time queries

Let i, j be a query (Notice: endpoints are inside blocks).

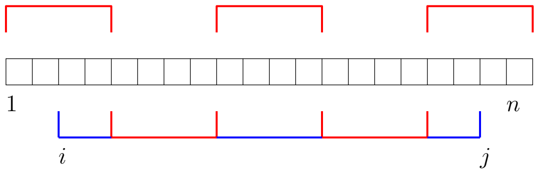


- We do not have the min in the prefix and suffix of every block.
- For each block, precompute it: $O(n)$ time and space!
- For the rest: use the structure we have built for A'.
- Now any such query takes $O(1)$ time.

Any problems?

$O(n)$ time and space and $O(\log n)$-time queries

Let i, j be a query (Notice: endpoints are inside blocks).

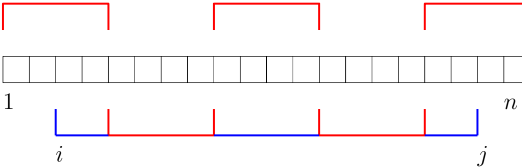


- We do not have the min in the prefix and suffix of every block.
- For each block, precompute it: $O(n)$ time and space!
- For the rest: use the structure we have built for A'.
- Now any such query takes $O(1)$ time.

Any problems? Yes!

$O(n)$ time and space and $O(\log n)$-time queries

Let i, j be a query (Notice: endpoints are inside blocks).

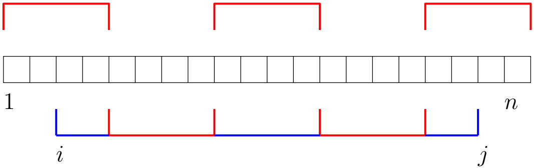


- We do not have the min in the prefix and suffix of every block.
- For each block, precompute it: $O(n)$ time and space!
- For the rest: use the structure we have built for A'.
- Now any such query takes $O(1)$ time.

Any problems? Yes! When i, j are fully contained in a single block.

$O(n)$ time and space and $O(\log n)$-time queries

Let i, j be a query (Notice: endpoints are inside blocks).



- We do not have the min in the prefix and suffix of every block.
- For each block, precompute it: $O(n)$ time and space!
- For the rest: use the structure we have built for A'.
- Now any such query takes $O(1)$ time.

Any problems? Yes! When i, j are fully contained in a single block.

Solution: Naïve search in block gives $O(\log n)$-time queries!

$O(n)$ time and space and $O(1)$-time queries

$O(n)$ time and space and $O(1)$-time queries

OK. Now it is time to show the breakthrough.

$O(n)$ time and space and $O(1)$-time queries

OK. Now it is time to show the breakthrough.

Theorem (Bender and Farach-Colton, LATIN 2000)

The RMQ problem can be solved in $O(1)$ *time after* $O(n)$ *time and space preprocessing.*

$O(n)$ time and space and $O(1)$-time queries

OK. Now it is time to show the breakthrough.

Theorem (Bender and Farach-Colton, LATIN 2000)

The RMQ problem can be solved in $O(1)$ time after $O(n)$ time and space preprocessing.

- We only have to deal with the strictly-inside-a-block case.

$O(n)$ time and space and $O(1)$-time queries

OK. Now it is time to show the breakthrough.

Theorem (Bender and Farach-Colton, LATIN 2000)

The RMQ problem can be solved in $O(1)$ time after $O(n)$ time and space preprocessing.

- We only have to deal with the strictly-inside-a-block case.
- We will show how to do that for a very restricted case:

$$|A[i+1] - A[i]| = 1$$

$O(n)$ time and space and $O(1)$-time queries

OK. Now it is time to show the breakthrough.

Theorem (Bender and Farach-Colton, LATIN 2000)

The RMQ problem can be solved in $O(1)$ time after $O(n)$ time and space preprocessing.

- We only have to deal with the strictly-inside-a-block case.
- We will show how to do that for a very restricted case:

$$|A[i+1] - A[i]| = 1$$

- We will then explain why this restricted case is sufficient!

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \ldots k]$, $Y[1 \ldots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \dots k]$, $Y[1 \dots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Normalize blocks by subtracting the initial offset from every element.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \dots k]$, $Y[1 \dots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Normalize blocks by subtracting the initial offset from every element.
- Choose $b = 1/2 \cdot \log n$ (instead of $b = \log n$).

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \dots k]$, $Y[1 \dots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Normalize blocks by subtracting the initial offset from every element.
- Choose $b = 1/2 \cdot \log n$ (instead of $b = \log n$).
- A normalized block is a vector of length $b - 1$.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \dots k]$, $Y[1 \dots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Normalize blocks by subtracting the initial offset from every element.
- Choose $b = 1/2 \cdot \log n$ (instead of $b = \log n$).
- A normalized block is a vector of length $b - 1$.
- There are $2^{b-1} = \Theta(\sqrt{n})$ distinct vectors.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \ldots k]$, $Y[1 \ldots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Normalize blocks by subtracting the initial offset from every element.
- Choose $b = 1/2 \cdot \log n$ (instead of $b = \log n$).
- A normalized block is a vector of length $b - 1$.
- There are $2^{b-1} = \Theta(\sqrt{n})$ distinct vectors.
- Precompute and store all answers!

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \ldots k]$, $Y[1 \ldots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \dots k]$, $Y[1 \dots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Create $\Theta(\sqrt{n})$ tables.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \dots k]$, $Y[1 \dots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Create $\Theta(\sqrt{n})$ tables.
- In each table put all $(\log n/2)^2$ answers to all in-block queries.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \ldots k]$, $Y[1 \ldots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Create $\Theta(\sqrt{n})$ tables.
- In each table put all $(\log n/2)^2$ answers to all in-block queries.
- $\Theta(\sqrt{n}\log^2 n)$ extra preprocessing.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \ldots k]$, $Y[1 \ldots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Create $\Theta(\sqrt{n})$ tables.
- In each table put all $(\log n/2)^2$ answers to all in-block queries.
- $\Theta(\sqrt{n} \log^2 n)$ extra preprocessing.
- Query time is $O(1)$.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \dots k]$, $Y[1 \dots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Create $\Theta(\sqrt{n})$ tables.
- In each table put all $(\log n/2)^2$ answers to all in-block queries.
- $\Theta(\sqrt{n} \log^2 n)$ extra preprocessing.
- Query time is $O(1)$.
- Preprocessing time and space is $O(n)$.

$O(n)$ time and space and $O(1)$-time queries

Assume: $|A[i+1] - A[i]| = 1$.

Observation: If two arrays $X[1 \dots k]$, $Y[1 \dots k]$ differ by some fixed value at each position; i.e. there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y.

- Create $\Theta(\sqrt{n})$ tables.
- In each table put all $(\log n/2)^2$ answers to all in-block queries.
- $\Theta(\sqrt{n} \log^2 n)$ extra preprocessing.
- Query time is $O(1)$.
- Preprocessing time and space is $O(n)$.

We still need to explain why this restricted case is sufficient!

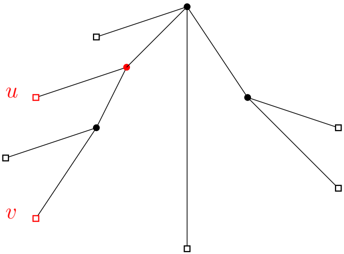
Lowest Common Ancestor queries

Lowest Common Ancestor queries

The lowest common ancestor (LCA) of two nodes u and v is the deepest node that is an ancestor of both u and v.

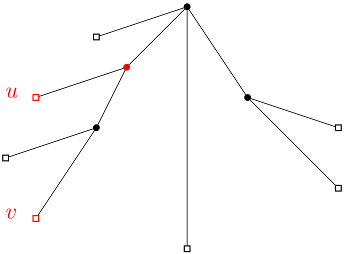
Lowest Common Ancestor queries

The lowest common ancestor (LCA) of two nodes *u* and *v* is the deepest node that is an ancestor of both *u* and *v*.



Lowest Common Ancestor queries

The lowest common ancestor (LCA) of two nodes u and v is the deepest node that is an ancestor of both u and v.



Definition

Given a rooted tree T, preprocess T so that the LCA of u and v can be computed efficiently.

Linear-Time Reduction: RMQ to LCA

Linear-Time Reduction: RMQ to LCA

Given array A we can build the **Cartesian tree** T of A:

Linear-Time Reduction: RMQ to LCA

Given array A we can build the **Cartesian tree** T of A:

- The root is a minimum element.

Linear-Time Reduction: RMQ to LCA

Given array A we can build the **Cartesian tree** T of A:

- The root is a minimum element.
- Removing this element splits the array into two.

Linear-Time Reduction: RMQ to LCA

Given array A we can build the **Cartesian tree** T of A:

- The root is a minimum element.
- Removing this element splits the array into two.
- Recurse on the two subarrays.

Linear-Time Reduction: RMQ to LCA

Given array A we can build the **Cartesian tree** T of A:

- The root is a minimum element.
- Removing this element splits the array into two.
- Recurse on the two subarrays.

Querying:

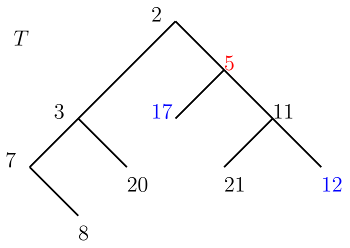
Linear-Time Reduction: RMQ to LCA

Given array A we can build the **Cartesian tree** T of A:

- The root is a minimum element.
- Removing this element splits the array into two.
- Recurse on the two subarrays.

Querying:

$$A = [8, 7, 3, 20, 2, \overset{i}{17}, 5, 21, 11, \overset{j}{12}]$$



Linear-Time Reduction: LCA to RMQ

Linear-Time Reduction: LCA to RMQ

Given a rooted tree T on n nodes we construct:

Linear-Time Reduction: LCA to RMQ

Given a rooted tree T on n nodes we construct:

- $E[1 \ldots 2n-1]$: node labels traversed in an Euler Tour (DFS).

Linear-Time Reduction: LCA to RMQ

Given a rooted tree T on n nodes we construct:

- $E[1 \ldots 2n-1]$: node labels traversed in an Euler Tour (DFS).
- $L[1 \ldots 2n-1]$: the distance of $E[i]$ from the root.

Linear-Time Reduction: LCA to RMQ

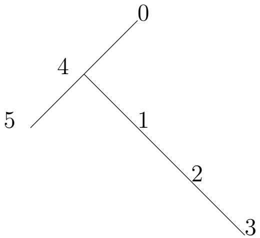
Given a rooted tree T on n nodes we construct:

- $E[1 \ldots 2n-1]$: node labels traversed in an Euler Tour (DFS).
- $L[1 \ldots 2n-1]$: the distance of $E[i]$ from the root.
- $R[i] = \min\{j : E[j] = i\}$ (first occurrence of $E[i]$ in the tour).

Linear-Time Reduction: LCA to RMQ

Given a rooted tree T on n nodes we construct:

- $E[1 \dots 2n-1]$: node labels traversed in an Euler Tour (DFS).
- $L[1 \dots 2n-1]$: the distance of $E[i]$ from the root.
- $R[i] = \min\{j : E[j] = i\}$ (first occurrence of $E[i]$ in the tour).



$$R = [1, 5, 6, 7, 2, 3]$$

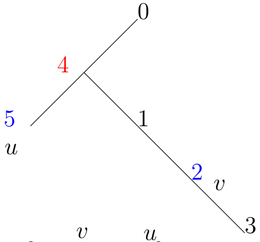
$$L = [0, 1, 2, 1, 2, 3, 4, 3, 2, 1, 0]$$

$$E = [0, 4, 5, 4, 1, 2, 3, 2, 1, 4, 0]$$

Linear-Time Reduction: LCA to RMQ

Querying:

- LCA(u, v) is translated to $E[\mathrm{RMQ}_L(R[u], R[v])]$.
- LCA$(5, 2)$ is
  $E[\mathrm{RMQ}_L(R[u], R[v])] = E[\mathrm{RMQ}_L(3, 6)] = E[4] = 4$.

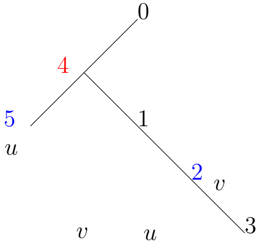


$$R = [1, 5, \overset{v}{6}, 7, 2, \overset{u}{3}]$$

$$L = [0, 1, \underset{i}{2}, 1, 2, \underset{j}{3}, 4, 3, 2, 1, 0]$$

$$E = [0, 4, 5, 4, 1, 2, 3, 2, 1, 4, 0]$$

Linear-Time Reduction: LCA to RMQ



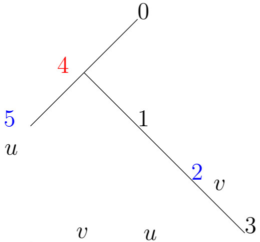
$$R = [1, 5, \overset{v}{6}, 7, 2, \overset{u}{3}]$$

$$L = [0, 1, \underset{i}{2}, 1, 2, \underset{j}{3}, 4, 3, 2, 1, 0]$$

$$E = [0, 4, 5, 4, 1, 2, 3, 2, 1, 4, 0]$$

Any observation about array L?

Linear-Time Reduction: LCA to RMQ



$$R = [1, 5, \overset{v}{6}, 7, 2, \overset{u}{3}]$$

$$L = [0, 1, \underset{i}{2}, {\color{red}1}, 2, \underset{j}{3}, 4, 3, 2, 1, 0]$$

$$E = [0, 4, 5, {\color{red}4}, 1, 2, 3, 2, 1, 4, 0]$$

Any observation about array L?

Yes: $|L[i+1] - L[i]| = 1$.

$O(n)$ time and space and $O(1)$-time queries

$O(n)$ time and space and $O(1)$-time queries

Let's put things together:

$O(n)$ time and space and $O(1)$-time queries

Let's put things together:

- Reduce RMQ problem to the LCA problem.

$O(n)$ time and space and $O(1)$-time queries

Let's put things together:

- Reduce RMQ problem to the LCA problem.
- Reduce LCA problem back to the RMQ problem.

$O(n)$ time and space and $O(1)$-time queries

Let's put things together:

- Reduce RMQ problem to the LCA problem.
- Reduce LCA problem back to the RMQ problem.
- **The reduction leads to a restricted RMQ problem**.

$O(n)$ time and space and $O(1)$-time queries

Let's put things together:

- Reduce RMQ problem to the LCA problem.
- Reduce LCA problem back to the RMQ problem.
- **The reduction leads to a restricted RMQ problem**.
- Solve it and we are done!

$O(n)$ time and space and $O(1)$-time queries

Let's put things together:

- Reduce RMQ problem to the LCA problem.
- Reduce LCA problem back to the RMQ problem.
- **The reduction leads to a restricted RMQ problem**.
- Solve it and we are done!

Theorem (Bender and Farach-Colton, LATIN 2000)

The RMQ problem can be solved in $O(1)$ time after $O(n)$ time and space preprocessing.

$O(n)$ time and space and $O(1)$-time queries

Let's put things together:

- Reduce RMQ problem to the LCA problem.
- Reduce LCA problem back to the RMQ problem.
- **The reduction leads to a restricted RMQ problem**.
- Solve it and we are done!

Theorem (Bender and Farach-Colton, LATIN 2000)

The RMQ problem can be solved in $O(1)$ time after $O(n)$ time and space preprocessing.

Theorem (Bender and Farach-Colton, LATIN 2000)

The LCA problem can be solved in $O(1)$ time after $O(n)$ time and space preprocessing.

Note

Lecture by P. Charalampopoulos. I slightly edited the slides, so I am responsible for any errors in them.