The LCA Problem Revisited

Michael A. Bender  
SUNY at Stony Brook

Martin 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: $\text{LCA}(e,f) = a$
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: \( \text{LCA}(e, f) = a \)
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, Golil, Schieber, Vishkin 89].

Remove PRAM complications ⇒ algorithm is sleek and sequential.
Naïve Solution: $<O(n^2), O(1)>$

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.

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

$\Rightarrow \langle O(n^2), O(1) \rangle$
Range Minimum Queries (RMQ)

Given an array \( A[1...n] \), \( \text{RMQ}[i,j] \) returns the index of the smallest element between \( i \) and \( j \).

\[
\begin{array}{cccccc}
  1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
  17 & 13 & 15 & 10 & 16 & 11 & 12 \\
\end{array}
\]

\( \text{RMQ}[2,5] = 4 \) because \( A[4] = 10 \) is min value in range.
Range Minimum Queries (RMQ)

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

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


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

- complexity measure: $\langle$ preprocess time, query time $\rangle$
Naive Solution for RMQ: \( O(n^2), O(1) \)

- 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 naive LCA and naive 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.

Euler tour E: CBCCAEADFDAC

Depth of nodes D: 010121123210
Reduction from LCA to RMQ

Use Euler Tour/DFS to convert LCA to RMQ.

Euler tour E

Depth of nodes D

\[ LCA[E, F] = A \]

Find first locations of \( E \) and \( F \) in Euler tour. RMQ between these locations in Depth Array \( \Rightarrow A \).
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 \(O(n^2), O(1)\) solution in stages.
\( \mathcal{O}(n \log n) \) Preprocessing

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

E.g., for \( i=1 \ldots n \) and \( l=0 \ldots \lfloor \log n \rfloor \), store \( \text{RMQ}[i, i+2^l] \).
**Preprocessing**

**O(n log n)**

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 $\text{RMQ}[i, i+2^\ell]$.

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.

\[ i \quad i+2^l \quad j \quad j-2^l \]

\( \Rightarrow O(1) \) queries.
Towards a $O(n), O(1)$ Algorithm

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

±1 RMQ. All neighbors differ by ±1.
Towards $O(n), O(1)$

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 $O(n), O(1)$

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 \( O\left(\frac{n}{\log n}\right) \) array:

\[
O\left(\frac{n}{\log n} \cdot \log\left(\frac{n}{\log n}\right)\right) = O(n)
\]
Towards $\langle O(n) \rangle O(1) \rangle$

- 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 $O(n) \circ O(1)$

- preprocessing for $O\left(\frac{n}{\log n}\right)$ array:
  
  $$O\left(\frac{n}{\log n} \cdot \log\left(\frac{n}{\log n}\right)\right) = O(n)$$

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

$\Rightarrow$ Closer to $O(n)$ but not there yet!
Improving $\pm 1$ RMQ in Small Arrays

Use $\pm 1$ structure! RMQ problem completed determined by pattern of $+1$'s and $-1$'s.

$log n \over 2$ array:

```
+1  -1  -1  +1  -1
```
Improving ±1 RMQ in Small Arrays

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

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

\[ +1 \ -1 \ -1 \ +1 \ -1 \]

$$\Rightarrow \text{only} \ 2^{\frac{1}{2} \log n - 1} = \frac{1}{2} \sqrt{n} \text{ distinct RMQ problems.}$$
Improving \( \pm 1 \) RMQ in Small Arrays

Use \( \pm 1 \) structure! RMQ problem completed determined by pattern of \( \pm 1 \)'s and \(-1\)'s.

\[
\frac{\log n}{2} \quad \text{array:} \quad \begin{array}{cccccc}
+1 & -1 & -1 & +1 & -1
\end{array}
\]

\[\Rightarrow \text{only} \quad 2^{\frac{1}{2} \log n - 1} = \frac{1}{2} \sqrt{n} \text{ distinct RMQ problems.}\]

Precompute all possible small RMQ problems in

\[O(\sqrt{n}) \cdot O(\log^2 n) = O(\sqrt{n} \log^2 n).\]
\[ O(n), O(1) \] LCA/\pm 1 \text{ RMQ}

\[
\frac{2n}{\log n} \rightarrow \frac{\log n}{\log n}
\]

\[ O(n) + O(\sqrt{n \log^2 n}) = O(n) \] 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: CBCCAEADFDAC

Depth of Nodes D: 0 1 0 1 2 1 2 3 2 1 0

Representative R: 421758
(first time node appears in DFS)

LCA(x, y) = E[RMQ_{depth_D}(R[x], R[y])]
Arbitrary RMQ

\[ \text{RMQ} \]

\[ \text{LCA} \quad \text{O}(n) \text{ reduction using Cartesian Trees.} \]

\[ \pm 1 \text{ RMQ} \quad \text{O}(n) \text{ reduction using Euler Tour.} \]

\[ \langle O(n), O(1) \rangle \]