Lowest Common Ancestor
(with a bit on on Range Minimum Queries)

Benjamin Sach

Preprocessing Summary
1. Construct \( N \) and \( D \) from \( T \)
2. Add a pointer from each node \( i \) to some \( N[j'] = i \)
3. Preprocess \( D \) for RMQs

Query Summary - \( \text{LCA}(i,j) \)
1. Find \( (\text{any}) \) \( x' \) at \( N[x'] = i \)
2. Find \( (\text{any}) \) \( x'' \) at \( N[x''] = j \)
3. Compute \( \text{RMQ}(x',x'') \) in \( D \)
4. \( \text{LCA}(i,j) = \text{RMQ}(x',x'') \)

Query time \( O(1+ \text{queryRMQ}(n)) \)

Preprocessing time \( O(n + \text{spaceRMQ}(n)) \)

Space \( O(n \log \log n) \)
We have seen an \(O(n \log n)\) space, \(O(n \log n)\) prep. time and \(O(1)\) query time solution for the Lowest Common Ancestor problem which uses solution 3 for RMQ from last lecture.

---

**Solving LCA using RMQ - correctness**

Claim the RMQ reports the location of some \(y\) in \(N\) at \(\text{LCA}(i, j) = y\).

\(i'\) and \(j'\) are in here so RMQ does not return the location of a \(y\) (all of the \(y\)'s are out of range).

---

**Solving LCA using RMQ - correctness**

We can also define a Euler tour of \(T\) recursively...

---

**Solving LCA using RMQ - correctness**

Notice anything interesting about \(D[i, j]\)?

\[D[i + 1] = D[i] \pm 1\]
**±1 Range minimum query**

Preprocess an integer array $A$ (length $n$) to answer range minimum queries:

where for all $k$, we have $A[k+1] = A[k] ± 1$

$$\begin{array}{cccccccccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

After preprocessing, a range minimum query is given by $\text{RMQ}(i, j) = k$

the output is the location of the smallest element in $A[i, j]$

(\text{in a tie, report the leftmost})

e.g. $\text{RMQ}(3, 7) = \text{null}$, which is the location of the smallest element in $A[3, 7]$

- Can we exploit this $±1$ property to get a more efficient RMQ data structure?
- Ideally we would like $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time

**Low-resolution RMQ (again)**

Key idea replace $A$ with a smaller, 'low resolution' array $H$

and many small arrays $L_0, L_1, L_2, \ldots$ for the details

Preprocess the array $H$ (which has length $n = 2m$) to answer $\text{RMQ}s$...

how do we answer a query in $A$ in $O(1)$ time?

Do one query in $H$ and one query in two different $L_i$ and return the smallest

**Preprocess each array $L_i$ (which has length $(\log (n)) / 2$) to answer $\text{RMQ}s$...**

too big and slow!... in $O(\log n \log \log n)$ space/prep time

as there are $O(n / \log n)$ $L_i$, arrays, we have $O(\log \log n)$ total space/prep time

Counting $±1$ RMQ arrays

How many different $±1$ RMQ arrays like this... are there?

We say that $L_x$ is equivalent to $L_y$ if for all $(i, j)$: $\text{RMQ}_x(i, j) = \text{RMQ}_y(i, j)$

(remember these are the locations of the minimum)

$$\begin{array}{cccccccccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Counting $±1$ RMQ arrays

How many different $±1$ RMQ arrays like this... are there?

We say that $L_x$ is equivalent to $L_y$ if for all $(i, j)$: $\text{RMQ}_x(i, j) = \text{RMQ}_y(i, j)$

(remember these are the locations of the minimum)

$$\begin{array}{cccccccccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Counting $±1$ RMQ arrays

How many different $±1$ RMQ arrays like this... are there?

We say that $L_x$ is equivalent to $L_y$ if for all $(i, j)$: $\text{RMQ}_x(i, j) = \text{RMQ}_y(i, j)$

(remember these are the locations of the minimum)

$$\begin{array}{cccccccccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Counting $±1$ RMQ arrays

How many different $±1$ RMQ arrays like this... are there?

We say that $L_x$ is equivalent to $L_y$ if for all $(i, j)$: $\text{RMQ}_x(i, j) = \text{RMQ}_y(i, j)$

(remember these are the locations of the minimum)

$$\begin{array}{cccccccccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

- We can precompute $d_x$ for each $L_x$ in $O(|L_x|) = O(\log n)$ time.
- How many different values of $d$ are there?
  - $d$ contains $(\log n) / 2 - 1$ bits so... at most $2^{(\log n) / 2} = (2^{\log n})^{1/2} \leq \sqrt{n}$
  - For each value of $d$ we store $\text{RMQ}(i, j)$ for all $i, j$
    - ...this requires $O(\sqrt{n} \log^2 n) = O(n)$ total space and prep. time
Key Idea: replace \( A \) with a smaller, ‘low resolution’ array \( H \)

Precompute all the RMQ answers for \( \mathcal{L} \) in \( O(n) \) total space and prep. time

To perform a query within some \( \mathcal{L} \):
- Look up \( d_x \)
- Find the row \( d_x \) in the table
- Find the entry giving \( \text{RMQ}_x(i,j) \)

This takes \( O(1) \) time

Ongoing Summary

We have seen an \( O(n \log \log n) \) space, \( O(n \log \log n) \) prep. time and \( O(1) \) query time solution for the Lowest Common Ancestor problem which uses solution 3 for RMQ from last lecture

We have seen an \( O(n) \) space, \( O(n) \) prep. time and \( O(1) \) query time solution for the \( \pm 1 \) Range Minimum Query problem which improves solution 3 for RMQ from last lecture (but only for \( \pm 1 \) inputs)

Checking the entries for \( \pm 1 \) RMQ

Solving LCAs using RMQs

Preprocessing Summary

\( \mathcal{O}(n) \) 1. Construct \( N \) and \( D \) from \( T \)
\( \mathcal{O}(n) \) 2. Add a pointer from each node \( i \) to some \( N[i'] = i \)
\( \mathcal{O}(n) \) 3. Preprocess \( D \) for RMQs

Query Summary - LCAs

\( \mathcal{O}(1) \) 1. Find \( (x,y) \) in \( N_i \) s.t. \( N_i[x'] = i \)
\( \mathcal{O}(1) \) 2. Find \( (x',y') \) in \( N_j \) s.t. \( N_j[y'] = j \)
\( \mathcal{O}(1) \) 3. Compute \( \text{RMQ}(x',y') \) in \( D \)
\( \mathcal{O}(1) \) 4. \( \text{LCA}(i,j) = N_i[\text{RMQ}(x',y')] \)

This gives us \( O(n) \) space, \( O(n) \) prep. time and \( O(1) \) query time for the LCA problem by using the solution to \( \pm 1 \) RMQ

Ongoing Summary

We have seen an \( O(n) \) space, \( O(n) \) prep. time and \( O(1) \) query time solution for the \( \pm 1 \) Range Minimum Query problem which improves solution 3 for RMQ from last lecture (but only for \( \pm 1 \) inputs)

We have seen an \( O(n) \) space, \( O(n) \) prep. time and \( O(1) \) query time solution for the Lowest Common Ancestor problem which uses the solution to \( \pm 1 \) RMQ

What about the general Range Minimum Query problem? (when the inputs aren’t \( \pm 1 \))

Solving RMQs using LCAs

Build the Cartesian tree, \( T_A \), of the array \( A \):
- The root is the smallest value
- The selected location partitions the array in two
- The root of the tree is given by recursively left and right

This process isn’t very efficient, a better one takes \( O(n) \) time

It’s not tricky but we don’t have time to cover it

Key Fact: The LCA in \( T_A \) equals the RMQ in \( A \)

This gives us \( O(n) \) space, \( O(n) \) prep. time and \( O(1) \) query time for the RMQ problem by using the solution to LCA :)

Optimal \( \pm 1 \) RMQ

Key Idea: replace \( A \) with a smaller, ‘low resolution’ array \( H \) and many small arrays \( L_0, L_1, L_2 \ldots \) for the \( \pm 1 \) RMQs

Preprocess each array \( L_x \) which has length \( \log \log n \) to answer RMQs...as there are \( O(n/\log n) \) \( L_x \) arrays, we have \( O(n) \) total space/prep time

How do we answer a query in \( A \) in \( O(1) \) times?

Do one query in \( H \) and one query in two different \( L_x \) and return the smallest
Summary

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the $\pm 1$ Range Minimum Query problem

which improves solution 3 for RMQ, from last lecture

(but only for $\pm 1$ inputs)

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the Lowest Common Ancestor problem

which uses the solution to $\pm 1$RMQ

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the Range Minimum Query problem

which uses the solution to LCA

(which works for all inputs)