Today: Constant-time tree queries
- range minimum queries
- lowest common ancestor
- level ancestors

Range Minimum Query (RMQ):
- preprocess array $A$ of $n$ numbers
- query: $RMQ(i, j) = \arg\min_k \{A[i], A[i+1], \ldots, A[j]\}$
  $= k$, $i \leq k \leq j$, minimizing $A[k]$

Lowest Common Ancestor (LCA):
- preprocess tree $T$ on $n$ nodes
- query: $LCA(x, y)$

Level Ancestors: (LA)
- preprocess tree $T$ on $n$ nodes
- query: $LA(x, k) = \text{parent}^k(x)$

Goal: $O(1)$ time/query, $O(n)$ space
[$\Theta(n^2)$ space trivial: store all answers]

Which of these problems are most similar? actually RMQ & LCA
Cartesian tree: [Gabow, Bentley, Tarjan - STOC 1984]

- reduction from array $A$ to binary tree $T$
- root of $T = \text{some min. element } A[i]$ in $A$
- left subtree = Cartesian tree of $A[<i]$
- right subtree = Cartesian tree of $A[>i]$

$A = \{8, 7, 2, 8, 6, 9, 4, 5\}$

- $T$ is a min heap
- in-order traversal of $T = A$
- $\text{LCA}(i, j) = \text{RMQ}(i, j)$

Linear-time construction algorithm:
- for each item in $A$: insert into $T$ by walking up right spine of $T$ & updating edge:

  $\text{insert}(3)$

  $O(1)$ changes

- charge walk to decrease in right spine len

  $\Rightarrow O(n)$ time (as in L14) [GBT84]
Reverse reduction: from (binary) tree $T$ to array $A$
- in-order traversal of $T$
- write depth of each node

$$\begin{array}{c}
\text{T} \\
/ \quad / \\
1 \quad 2 \\
\quad / \\
2 \quad 1 \\
\quad / \\
3 \quad 3
\end{array}$$

$$\Rightarrow 2 \ 1 \ \varnothing \ 3 \ 2 \ 3 \ 1 \ 2$$

- $\text{RMQ}(i, j) = \text{LCA}(i, j)$
  $\uparrow$ index into $A$ $\uparrow$ node in $T$

$\text{RMQ universe reduction:}$
- reduce $\text{RMQ} \rightarrow \text{LCA} \rightarrow \text{RMQ}$
  Cartesian in-order depth
- $\text{RMQ}(i, j)$ answers are preserved $\uparrow$ indices in array (argmin)
- arbitrary ordered universe $\rightarrow \{0, 1, \ldots, n-1\}$
- $O(n)$ time in comparison model
Constant-time $LCA \Rightarrow RMQ$: [Harel & Tarjan - SICOMP 1984]
- simplified by [Bender & Farach-Colton - LATIN 2000]*
- based on PRAM [Berkman et al. - STOC 1989]

1. reduce to $\pm 1$ RMQ: adjacent values differ by $\pm 1$
   - Euler tour of tree (depth-first search),
     writing depth of each node visited
     (instead of in-order traversal)
   - e.g. $\emptyset 1 2 1 \emptyset 1 2 3 2 3 2 1 2 1 \emptyset$
     \[\text{root}\]
   \[\Rightarrow \pm 1: \text{also works for nonbinary trees}\]
   - each node stores its first (or any) visit
   - each visit stores corresponding node
   - $LCA(x, y) = RMQ(\text{first}(x), \text{first}(y))$

2. $O(1)$ time, $O(n \log n)$ space RMQ:
   choices:
   - store answer from every start point
     of interval of length $= \text{power of } 2$
   - any interval is the (nondisjoint) union
     of two such intervals:
     \[\text{length } k\]
     \[\Rightarrow \text{RMQ} = \text{(arg) min of } 2 \text{ stored answers}\]
3. Indirection: Split array into groups of \( \frac{1}{2} \lg n \)

\[
\begin{array}{c}
\min \\
\frac{1}{2} \lg n & \frac{1}{2} \lg n & \ldots & \frac{1}{2} \lg n \\
2n/\lg n & \text{query} & \text{on mins of groups}
\end{array}
\]

\[\Rightarrow \text{top is } O(1) \text{ time, } O(n) \text{ space} \]
- \( \text{RMQ}(i,j) = (\text{arg})\min \text{ of:} \)
  - \( \text{RMQ}(i,\infty) \text{ in } i \text{'s group} = \lfloor \frac{2^i}{\lg n} \rfloor \)
  - \( \text{RMQ}(-\infty,j) \text{ in } j \text{'s group} \)
  - \( \text{RMQ}(i \text{'s group} + 1, j \text{'s group} - 1) \text{ in top} \)

4. Lookup table for groups: \( (n' = \frac{1}{2} \lg n) \)
- Add \(-A[\emptyset]\) to every value \( \Rightarrow A'[\emptyset] = \emptyset \)
- \( \text{RMQ}(i,j) \) invariant under such shift
\[\Rightarrow \# \text{ possible } A' \text{ arrays} = \# \pm 1s = 2^{n'} = \sqrt{n} \]
- \( (\frac{1}{2} \lg n)^2 \) possible queries
- \( O(\lg \lg n) \) bits to store an answer
\[\Rightarrow \text{lookup table storing all answers} \]
  for all possible \( A' \) arrays
  uses \( O(\sqrt{n} \lg^2 n \lg \lg n) = o(n) \) bits
- Each group just stores index into table describing \( A' \) array \( \sim O(n) \) words
\[\Rightarrow O(1) \text{ query at bottom} \]
- Total: \( O(1) \) query, \( O(n) \) (words of) space
- \( O(n) \) bits for LCA & RMQ! [Sadakane-JDA 2007]
Constant-time level ancestors:
[Berkman & Vishkin - JCSS 1994; Dietz - WADS 1991; Alstrup & Holm - ICALP 2000; dynamic trees
Bender & Farach-Colton - TCS 2004] * HERE

1. **jump pointers**: $O(n \lg n)$ space, $O(\lg n)$ query
   - each node stores pointer to $2^i$th ancestor for $i = 0, 1, \ldots, \lg n$ (or less)
   - query: $x = 2^\lfloor \lg k \rfloor$th ancestor of $x$
     $k = k - 2^\lfloor \lg k \rfloor < k/2 \Rightarrow O(\lg n)$

2. **long-path decomposition**: $O(n)$ space, $O(\lg n)$ query
   - find longest root-to-leaf path (deepest leaf)
   - store nodes on path in depth-ordered array
     - each node stores array & index of itself
   - recurse on subtrees hanging off path
   - query: if $k \leq$ index $i$ of node $x$ in its path:
     return path array[$i - k$]
   else: $x = \text{parent}(\text{path array}[0])$
     $k = k - 1 - i$
     repeat

extra

- node of height $h$ is on path of length $\geq h$
- but can visit $O(\Omega h)$ paths:
3. **ladder decomposition**: $O(n)$ space, $O(lg n)$ query
   - extend each path upward into ladder of twice the length \(\Rightarrow\) ladders overlap
   - \(\leq\) double the space of \(\odot\)
   - node stores which ladder contains it in the lower half (corresp. to unique path)
   - ladder = array; query uses them as in \(\odot\)
   - node of height \(h\) is on ladder of height \(\geq 2h\)
   \(\Rightarrow\) each step at least doubles height of node

4. **Combine jump pointers \(\odot\) & ladder decomp. \(\odot\)**
   - over time: exp decr. hops \(\sim\) expr. incr. hops
   - query: 1 jump pointer \(\rightarrow\) height \(\geq \frac{k}{2}\) above \(\times\)
   - + 1 ladder step \(\rightarrow\) (ladder height \(\geq k\) above)
   \(\Rightarrow\) $O(1)$ query, $O(n \lg n)$ space

5. **tune jump pointers**: $O(n + L \lg n)$ space
   - each node stores a descendent leaf & how much deeper it is
   \(\Rightarrow\) can start query at a leaf \(k' = k + d\)
   \(\Rightarrow\) only need jump pointers at leaves
leaf trimming: (indirection) [Alstrup, Husfeldt, Rauhe 1995]
- cut below maximally deep nodes with \( \geq \frac{1}{4} \log n \) descendants
- \# leaves in top = \( O(n^{1/2 \log n}) \)
- \( n \) on top uses \( O(n) \) space
- query tries in bottom; else uses top

lookup table for bottom trees with \( n' < \frac{1}{4} \log n \)
- \# rooted trees on \( n' \) nodes = \( C_{n'} \leq 2^{\frac{n'}{2}} \)
- \# queries = \( (n')^2 = O(\log^2 n) \)
- answer = \( O(\log \log n) \)

\( \Rightarrow \) lookup table storing all answers for all possible trees uses \( O(\sqrt{n} \log^2 n \log \log n) = o(n) \) bits
- bottom tree stores index into table

\( \Rightarrow O(1) \) query, \( O(n) \) space!

Dynamic LCA: [Cole & Hariharan - SIComp 2005]
- \( O(1) \) updates:
  - insert/delete leaves
  - subdivide/merge edges

Dynamic LA: [Alstrup & Holm - ICALP 2000]
- insert leaves, & edges in a forest
- OR insert leaves & root, amortized [Dietz - WADS 1991]