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

Range Minimum Query (RMQ):
- preprocess array $A$ of $n$ numbers
- query: $RMQ(i, j) = (\text{arg}\min\{A[i], A[i+1], \ldots, A[j]\})$
  $= k, i \leq k \leq j$, 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 $T$ = some min. element $A[i]$ in $A$
  - Left subtree = Cartesian tree of $A[<i]$
  - Right subtree = Cartesian tree of $A[>i]$

- $T$ 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:

  - $\Rightarrow 0(1)$ changes

- Charge walk to decrease in right spine len

  $\Rightarrow 0(n)$ time (as in L14) [GBT84]

Seventeen in comparison model
Reverse reduction: from (binary) tree $T$ to array $A$
- in-order traversal of $T$
- write depth of each node

RMQ universe reduction:
- reduce $\text{RMQ} \to \text{LCA} \to \text{RMQ}$
- $\text{RMQ}(i, j)$ answers are preserved
- arbitrary ordered universe $\rightarrow \{0, 1, \ldots, n-1\}$
- $O(n)$ time in comparison model
Constant-time $\text{LCA} \Rightarrow \text{RMQ}$: \[\text{[Harel \& Tarjan - STOC 1984]}\]
- Simplified by \[\text{[Bender \& Farach-Colton - LATIN 2000]}\]
- Based on PRAM \[\text{[Berkman et al. - STOC 1989]}\]

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. $\emptyset 1 2 1 \emptyset 1 2 3 2 3 2 1 2 1 \emptyset$
     - Root
   \[\Rightarrow \pm 1; \text{ also works for nonbinary trees}\]
   - Each node stores its first (or any) visit
   - Each visit stores corresponding node
   - $\text{LCA}(x,y) = \text{RMQ}($first$(x), \text{first}(y))$

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

\[
\begin{array}{c}
\text{min} \quad 2n/\log n \\
\frac{1}{3} \log n \quad \frac{1}{3} \log n \quad \ldots \quad \frac{1}{3} \log n
\end{array}
\]

\[ \Rightarrow \text{on mins of groups} \]

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

4. Lookup table for groups: \( n' = \frac{1}{2} \log n \)
- add \(-A[\emptyset]\) to every value \( \Rightarrow A'[\emptyset] = \emptyset \)
- \( \text{RMQ}(i, j) \) invariant under such shift
\[ \Rightarrow \# \text{ possible } A' \text{ arrays} = \# \pm 1s = 2n' = \sqrt{n} \]
- \( (\frac{1}{2} \log n)^2 \) possible queries
- \( O(\log \log n) \) bits to store an answer
\[ \Rightarrow \text{lookup table storing all answers for all possible } A' \text{ arrays} \]
  - uses \( O(\sqrt{n} \log^2 n \log \log n) = o(n) \) 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: $O(n \log n)$ space, $O(\log n)$ query
- each node stores pointer to $2^i$th ancestor for $i = 0, 1, \ldots, \lfloor \log n \rfloor$ (or less)
- query: $x = 2^{\lfloor \log k \rfloor}$th ancestor of $x$
  \[ k = k - 2^{\lfloor \log k \rfloor} < k/2 \Rightarrow O(\log n) \]

② long-path decomposition: $O(n)$ space, $O(n)$ 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

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

extra

4. **combine jump pointers** ① & ladder decomp. ③ over time: exp. decr. hops $\sim$ expr. incr. hops
   - query: 1 jump pointer $\rightarrow$ height $\geq \frac{k}{2}$ above $\times$
     + 1 ladder step (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 – FOCS 97]
- cut below maximally deep nodes with $\geq \frac{3}{4} \log n$ descendants
- # leaves in top = $O\left(\frac{n}{\log n}\right)$
- # 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 = $C_{n'} \leq 2^{\frac{3}{2}n'}$< $\frac{4\log n}{\sqrt{n}}$
- # queries = $(n')^2 = O\left(\log^2 n\right)$
- answer = $O(\log \log n)$
- 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]