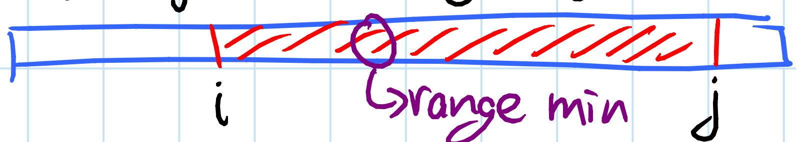
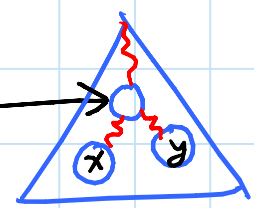
6.851	Lecture 15	Apr. 12, 2012

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

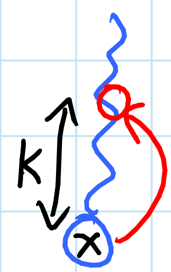
Range Minimum Query (RMQ):
- preprocess array A of n numbers
- query: $RMQ(i,j) = (\text{arg})\min\{A[i], A[i+1], \ldots, A[j]\}$
  $= k,\ i \le k \le j$, minimizing $A[k]$



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



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



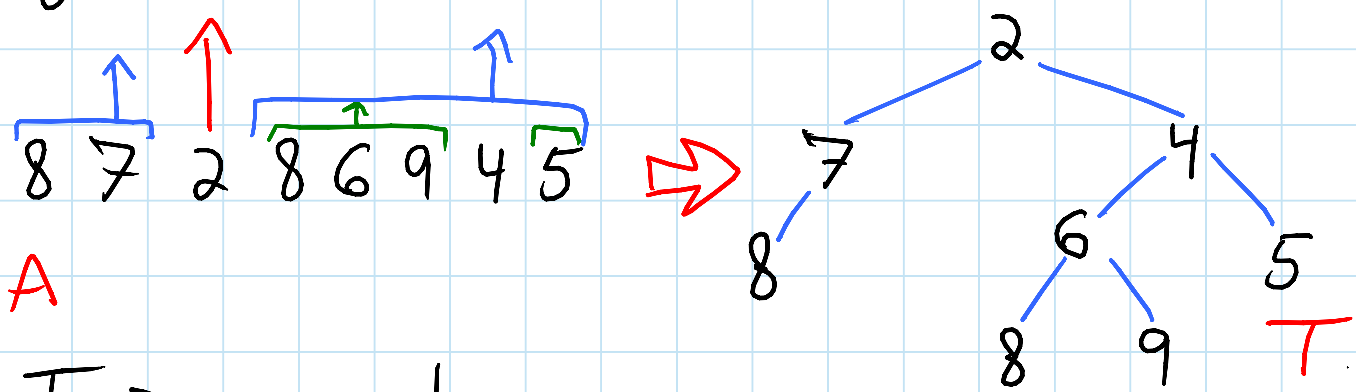
Goal: $O(1)$ time/query, $O(n)$ space
[$\Theta(n^2)$ space trivial: store all answers]

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]

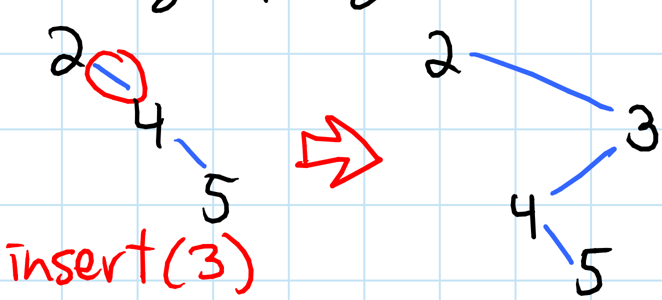


- T is a min heap
- in-order traversal of T = A
- LCA(i, j) = RMQ(i, j)

tree nodes   array indices

Linear-time construction algorithm:

- for each item in A: insert into T by walking up right spine of T & updating edge:



O(1) changes

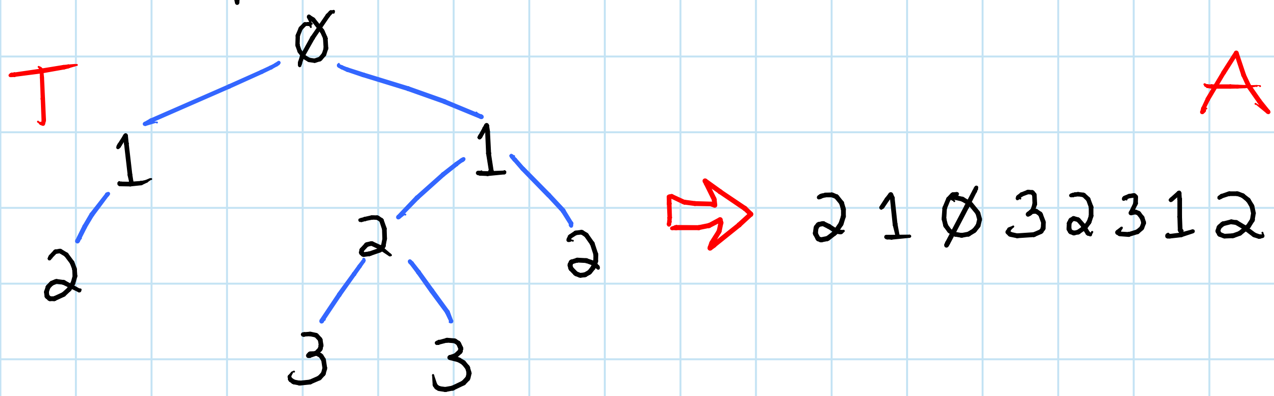
- charge walk to decrease in right spine len.

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

↳ even in comparison model

Reverse reduction: from (binary) tree T to array A

- in-order traversal of T
- write depth of each node



- $RMQ(i,j) = LCA(i,j)$
  - i,j in RMQ: index into A
  - i,j in LCA: node in T

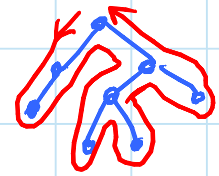
RMQ universe reduction:

- reduce RMQ $\xrightarrow{\text{Cartesian}}$ LCA $\xrightarrow{\text{in-order depth}}$ RMQ
- RMQ(i,j) answers are preserved
  - indices in array (argmin)
- arbitrary ordered universe $\to \{0, 1, \ldots, n-1\}$
- $O(n)$ time in comparison model

Constant-time LCA ⇒ RMQ: [Harel & Tarjan – SICOMP 1984]

- Simplified by [Bender & Farach-Colton – LATIN 2000]* ← HERE
- based on PRAM [Berkman et al. – STOC 1989]

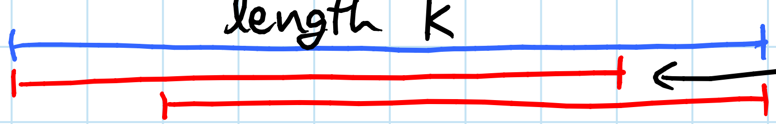
① reduce to ±1 RMQ: adjacent values differ by ±1
- Euler tour of tree (depth-first search), writing depth of each node visited (instead of in-order traversal)



- e.g. $\emptyset$ 1 2 1 $\emptyset$ 1 2 3 2 3 2 1 2 1 $\emptyset$ root

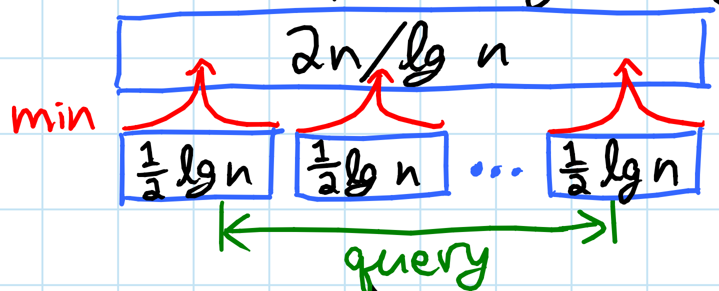
⇒ ±1; also works for nonbinary trees
- each node stores its first (or any) visit
- each visit stores corresponding node
- LCA(x, y) = RMQ(first(x), first(y))

② O(1) time, O(n lg n) space RMQ: — choices:
- store answer from every start point ← n
  of interval of length = power of 2 ← lg n
- any interval is the (nondisjoint) union of two such intervals:



⇒ RMQ = (arg) min of 2 stored answers

③ <u>indirection</u>: split array into groups of $\frac{1}{2}\lg n$



← ② on mins of groups

$\Rightarrow$ top is $O(1)$ time, $O(n)$ space

- RMQ(i, j) = (arg)min of:
  - RMQ(i, ∞) in i's group $= \lfloor 2i / \lg n \rfloor$
  - RMQ(−∞, j) in j's group
  - RMQ(i's group + 1, j's group − 1) in top

④ <u>lookup table for groups</u>: $(n' = \frac{1}{2}\lg n)$

- add $-A[\emptyset]$ to every value $\Rightarrow A'[\emptyset] = \emptyset$
  - RMQ(i, j) invariant under such shift

$\Rightarrow$ # possible A' arrays = # ±1s $= 2^{n'} = \sqrt{n}$

- $(\frac{1}{2}\lg n)^2$ possible queries
- $O(\lg\lg n)$ bits to store an answer

$\Rightarrow$ 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 ~ $O(n)$ words

$\Rightarrow$ $O(1)$ query at bottom

- <u>total</u>: $O(1)$ query, $O(n)$ (words of) space
- $O(n)$ <u>bits</u> for LCA & RMQ! [Sadakane – JDA 2007]

Constant-time level ancestors:

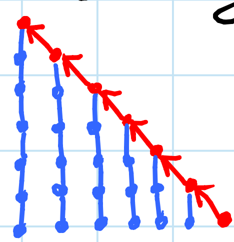
[Berkman & Vishkin – JCSS 1994; Dietz – WADS 1991; Alstrup & Holm – ICALP 2000; ⇐ dynamic trees
Bender & Farach-Colton – TCS 2004] * ⇐ HERE

① jump pointers: $O(n \lg n)$ space, $O(\lg n)$ query
- each node stores pointer to 2^ith ancestor for $i = 0, 1, \ldots, \lg n$ (or less)

extra
- query: $x = 2^{\lfloor \lg k \rfloor}$th ancestor of x
  $k = k - 2^{\lfloor \lg k \rfloor} < k/2 \Rightarrow O(\lg n)$
  repeat

② long-path decomposition: $O(n)$ space, $O(\sqrt{n})$ 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

extra
- query: if $k \le$ index i of node x in its path:
  return path array$[i-k]$
  else: x = parent(path array$[\emptyset]$)
  $k = k - 1 - i$
  repeat
- node of height h is on path of length $\ge h$
- but can visit $\Theta(\sqrt{n})$ paths:



③ <u>ladder decomposition</u>: $O(n)$ space, $O(\lg n)$ query

- extend each path upward into <u>ladder</u> of twice the length (⇒ ladders overlap)

⇒ ≤ double the space of ②

- node stores which ladder contains it in the lower half (corresp. to unique path)
- ladder = array; query uses them as in ②

extra
- node of height h is on ladder of height $\geq 2h$

⇒ each step at least doubles height of node

④ <u>combine jump pointers</u> ① & <u>ladder decomp.</u> ③

over time: exp. decr. hops ~ expr. incr. hops

- <u>query</u>: 1 jump pointer → height $\geq \frac{k}{2}$ above x
  + 1 ladder step (ladder height $\geq k$ above)

⇒ $O(1)$ query, $O(n \lg n)$ space

⑤ <u>tune jump pointers</u>: $O(\underbrace{n}_{\text{ladders}} + \underbrace{L \lg n}_{\text{jump pointers}})$ space

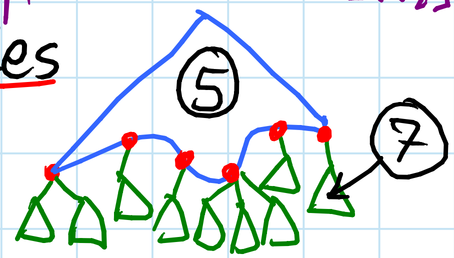
L = # leaves

- each node stores a descendent leaf & how much deeper d it is

⇒ can start query at a leaf ($k' = k + d$)

⇒ only need jump pointers at leaves

⑥ <u>leaf trimming</u>: (indirection) [Alstrup, Husfeldt, Rauhe - FOCS 1997]
- cut below maximally deep nodes with $\geq \frac{1}{4} \lg n$ descendants
$\Rightarrow$ # leaves in top $= O(n/\lg n)$
$\Rightarrow$ ⑤ on top uses $O(n)$ space
- query tries in bottom; else uses top



⑦ <u>lookup table</u> for bottom trees with $n' < \frac{1}{4} \lg n$
- # rooted trees on $\underbrace{n'}_{< \frac{1}{4}\lg n}$ nodes $= C_{n'} \leq \underbrace{2^{2n'}}_{< \sqrt{n}}$

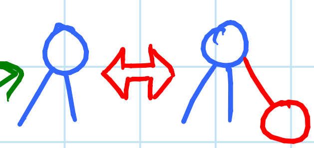
Catalan

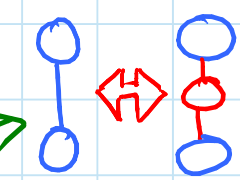
encoding proof: encode $2n'$ steps of Euler tour as up/down ($\emptyset$ / 1)

- # queries $= (n')^2 = O(\lg^2 n)$
- answer $= O(\lg \lg n)$
$\Rightarrow$ lookup table storing all answers for all possible trees uses $O(\sqrt{n} \lg^2 n \lg \lg n)$ $= o(n)$ bits.
- bottom tree stores index into table

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

<u>Dynamic LCA</u>: [Cole & Hariharan - SICOMP 2005]
- $O(1)$ updates:
  - insert/delete leaves
  - subdivide/merge edges





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