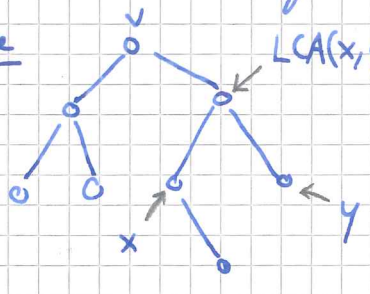
Advanced Data Structures: Lecture 2

Static Trees / Array Preprocessing

tree

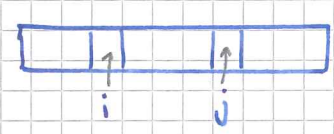


$LCA(x,y)$ = lowest common ancestor

$LA(x,k)$: $parent^{(k)}(x)$ go k levels up, level ancestry

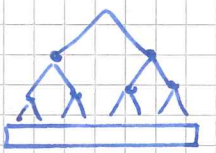
prepare for such queries so that we can answer in $O(1)$
→ make trees „behave" more like arrays

array



$RMQ(i,j) = \operatorname{argmin}\{x_i, x_{i+1}, \ldots, x_j\}$
range minimum query (return the index)

doable in $O(\log n)$ with a range/segment tree:
(works for any associative operation and can not be beaten in general (especially for Prefix Sum) but it can be improved for RMQ)

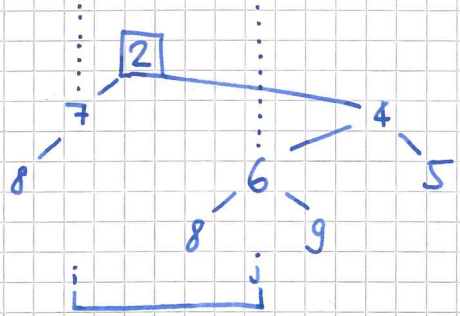


goal: $O(n)$ time/space preprocessing (in a Word-RAM-model, so $\Theta(n \log n)$ bits)
$O(1)$ time queries

RMQ → LCA (Gabow et al. 1984)

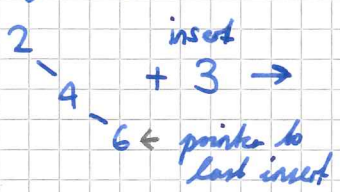
$A = [\,8\ \ 7\ \ \boxed{2}\ \ 8\ \ 6\ \ 9\ \ 4\ \ 5\,]$

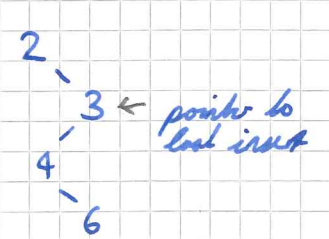
Idea: build Cartesian tree (break ties arbitrarily)



$RMQ(i,j) = LCA(i,j)$

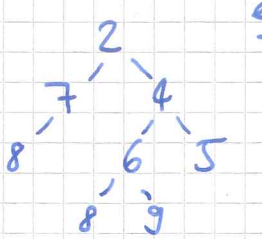
How to do it?
- $O(n^2)$ easy
- $O(n)$: go through it from left to right & keep track of the right spine:





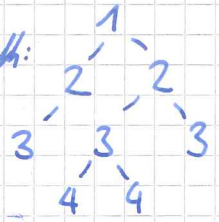
effort to walk upwards amortizes over all the insertions, so $O(1)$ per op.
→ $O(n)$ overall

LCA → RMQ



easy → in-order traversal: 8 7 2 8 6 9 4 5

almost as easy → relabel by depth:



→ 3 2 1 4 3 4 2 3

this reduces the universe size: $U = [n]$

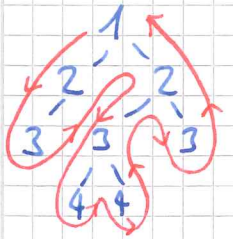
from this: RMQ → LCA → RMQ

U large $\hat{U} = [n]$

Taking this further Harel & Tarjan 1983, Bender & Farach Colton 2000

① tree → ±1 RMQ

$|A[i] - A[i+1]| = 1$



Euler tour traversal

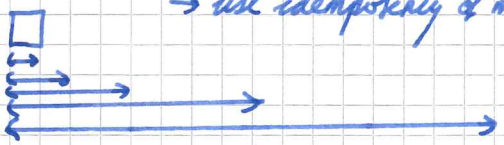
A = [1 2 3 2 1 2 3 4 3 4 3 2 3 2 1] still of size $O(n)$

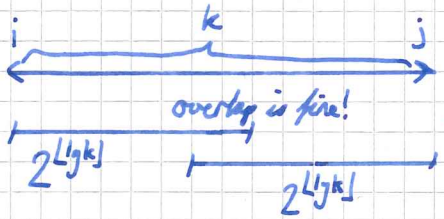
(each edge visited twice, so new A is of size $\leq 2n$)

② RMQ in $O(1)$ time but with $O(n \log n)$ preprocessing time

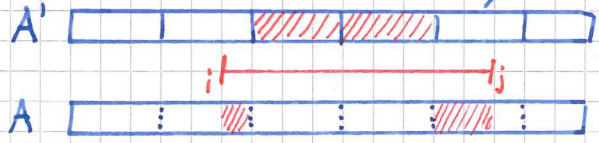
keep powers of 2 as precomputed answers

→ use idempotency of min:





③ Indirection: Do ② on a sequence of size $O(\frac{n}{\frac{1}{2}\log n})$



$\frac{1}{2}\log n$-sized blocks aggregated into single elements

decompose the query

④ precompute for small blocks:

…−1 +1 −1 +1 +1

initial value does not change the answer

$2^{\frac{1}{2}\log n} = \sqrt{n}$ different blocks (looking only at the +1, −1 pattern)

$(\frac{1}{2}\log n)^2$ queries

$\log \log n$ answers

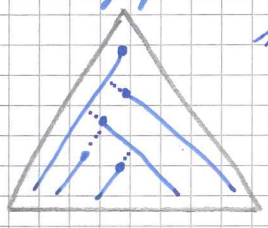
$O(\sqrt{n} \log^2 n \log \log n) = o(n)$ overall

level ancestry $LA(x,k)$ Dietz 1991, Berkman & Vishkin 1994, Bender & Farach-Colton '04
several ideas needed to do it in $O(1)$ per query

① **jump pointers**: store all 2^i-parents

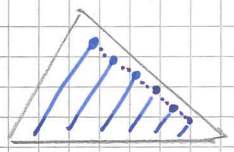
$O(n \log n)$ space & preprocessing time (induction over i)
$O(\log n)$ query

② **long path decomposition**



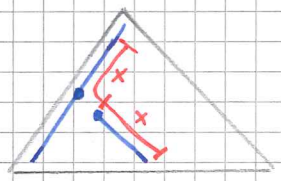
take longest root-leaf-path in the tree → store this path in an array
→ recurse in the remaining subtrees
for query: if array/path is not long enough, use parent pointers towards the root

Worst case: $\Theta(\sqrt{n})$ paths of length $\Theta(\sqrt{n})$



→ $O(n)$ space for $\Theta(\sqrt{n})$ query

③ **ladder decomposition**

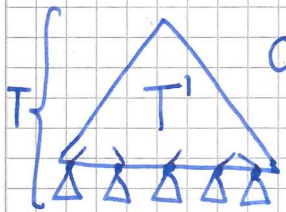


extend the paths from ② by the same length further upwards
queries get accelerated:
node height $\geq h$ → path length $\geq h$ → jump to $\geq 2h$

→ $O(n)$ space and $O(\log n)$ query

④ **combine ① and ③**
to jump k levels up from x: follow the jump pointer $\frac{k}{2} \leq 2^{\lfloor \log k \rfloor} \leq k$ to some y
now y has height $\geq \frac{k}{2}$ → a single ladder is enough
→ $O(n \log n)$ preprocessing and $O(1)$ query

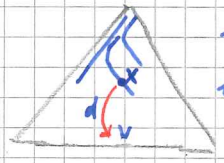
⑤ **Indirection: cut small trees**



$O(\frac{n}{\log n})$ leaves
$\leq \frac{1}{4}\log n$ nodes

#trees: $2^{2 \cdot \frac{1}{4}\log n} = \sqrt{n}$ (Catalan numbers) $o(n)$
#queries: $(\frac{1}{4}\log n)^2$
#answers: $\log\log n$

⑥ **jump pointers for leaves only** (use on T')



keep a leaf and its depth for each vertex: $LA(x,k) = LA(v, k+d)$

$O(n + L \cdot \log n)$ space for L leaves

⑤+⑥ combined:
$O(n)$ space and $O(1)$ query time