9 Range-Minimum Queries

27 April 2020

Sebastian Wild

version 2020-04-28 22:43 H

Outline

9 Range-Minimum Queries

9.1 Introduction

Range-minimum queries (RMQ)

array/numbers don't change

- **Given:** Static array $A[0..n)$ of numbers
- **Goal:** Find minimum in a range;
  A known in advance and can be preprocessed

RMQ(7, 15) = 10

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10	

- **Nitpicks:**
  - Report *index* of minimum, not its value
  - Report *leftmost* position in case of ties

Rules of the Game

- comparison-based $\rightsquigarrow$ values don't matter, only relative order

- Two main quantities of interest:
  1. **Preprocessing time**: Running time $P(n)$ of the preprocessing step
  2. **Query time**: Running time $Q(n)$ of one query (using precomputed data)

- Write "$\langle P(n), Q(n)\rangle$ time solution" for short

9.2 RMQ, LCP, LCE, LCA — WTF?

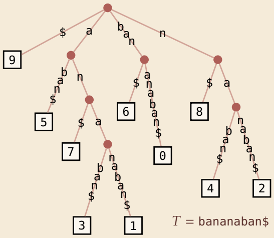
Recall Unit 6

Application 4: Longest Common Extensions

- We implicitly used a special case of a more general, versatile idea:

  The ***longest common extension (LCE)*** data structure:
  - **Given:** String $T[0..n-1]$
  - **Goal:** Answer LCE queries, i. e.,
    given positions i, j in T,
    how far can we read the same text from there?
    formally: $\mathrm{LCE}(i, j) = \max\{\ell : T[i..i+\ell) = T[j..j+\ell)\}$

$\leadsto$ use suffix tree of T!



longest common prefix of ith and jth suffix

- In $\mathcal{T}$: $\mathrm{LCE}(i, j) = \mathrm{LCP}(T_i, T_j) \leadsto$ same thing, different name!
  $=$ string depth of *lowest common ancester (LCA)* of leaves $\boxed{i}$ and $\boxed{j}$

- in short: $\boxed{\mathrm{LCE}(i, j) = \mathrm{LCP}(T_i, T_j) = \mathrm{stringDepth}\big(\mathrm{LCA}(\boxed{i}, \boxed{j})\big)}$

Recall Unit 6

Efficient LCA

How to find lowest common ancestors?

- Could walk up the tree to find LCA $\rightsquigarrow$ $\Theta(n)$ worst case
- Could store all LCAs in big table $\rightsquigarrow$ $\Theta(n^2)$ space and preprocessing

Amazing result: Can compute data structure in $\Theta(n)$ time and space that finds any LCA is **constant(!) time**.

- a bit tricky to understand
- but a theoretical breakthrough
- and useful in practice

and suffix tree construction inside . . .

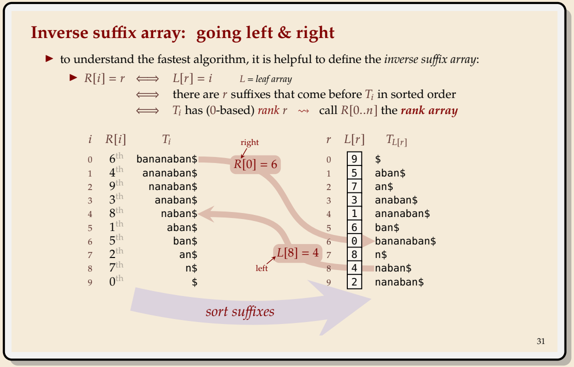
$\rightsquigarrow$ for now, use $O(1)$ LCA as black box.

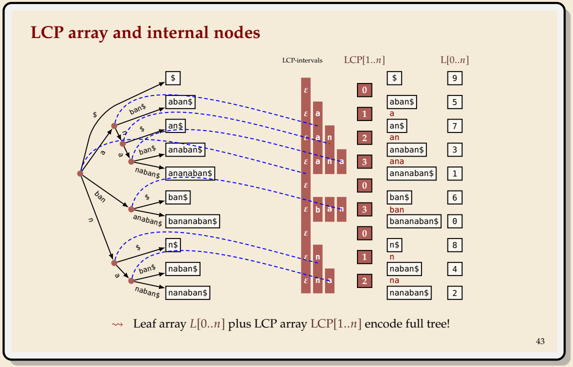
$\rightsquigarrow$ $\boxed{\text{After linear preprocessing (time \& space), we can find LCEs in } O(1) \text{ time.}}$

Finally: Longest common extensions

- In Unit 6: Left question open how to compute LCA in suffix trees
- But: Enhanced Suffix Array makes life easier!

$$\boxed{\mathrm{LCE}(i,j) \;=\; \mathrm{RMQ}_{\mathrm{LCP}}(R[i]+1, R[j])}$$





RMQ Implications for LCE

- Recall: Can compute (inverse) suffix array and LCP array in $O(n)$ time

$\rightsquigarrow$ A $\langle P(n), Q(n) \rangle$ time RMQ data structure implies a $\langle P(n), Q(n) \rangle$ time solution for longest-common extensions

9.3 Sparse Tables

Trivial Solutions

$\mathrm{RMQ}(7, 15) = 10$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

- Two easy solutions show extreme ends of scale:

1. **Scan on demand**
   - no preprocessing at all
   - answer $\mathrm{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min

   $\rightsquigarrow \langle O(1), O(n) \rangle$

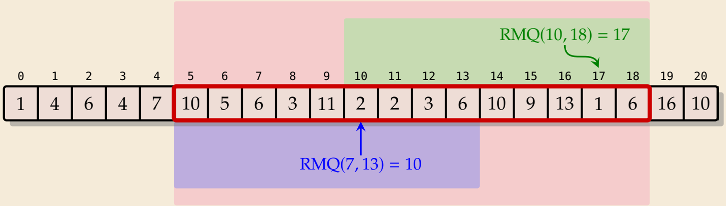
2. **Precompute all**
   - Precompute all answers in a big 2D array $M[0..n)[0..n)$
   - queries simple: $\mathrm{RMQ}(i, j) = M[i][j]$

   $\rightsquigarrow \langle O(n^3), O(1) \rangle$

   - Preprocessing can reuse partial results $\rightsquigarrow$ $\langle O(n^2), O(1) \rangle$

Sparse Table

- **Idea:** Like "precompute-all", but keep only some entries
- store $M[i][j]$ iff $\ell = j - i + 1$ is 2^k.
  $\rightsquigarrow \le n \cdot \lg n$ entries
- How to answer queries?

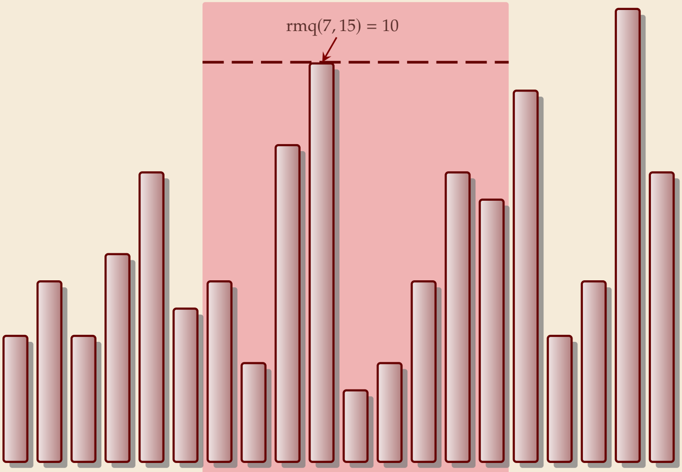


- Preprocessing can be done in $O(n \log n)$ times

$\rightsquigarrow$ $\langle O(n \log n), O(1) \rangle$ time solution!

9.4 Cartesian Trees

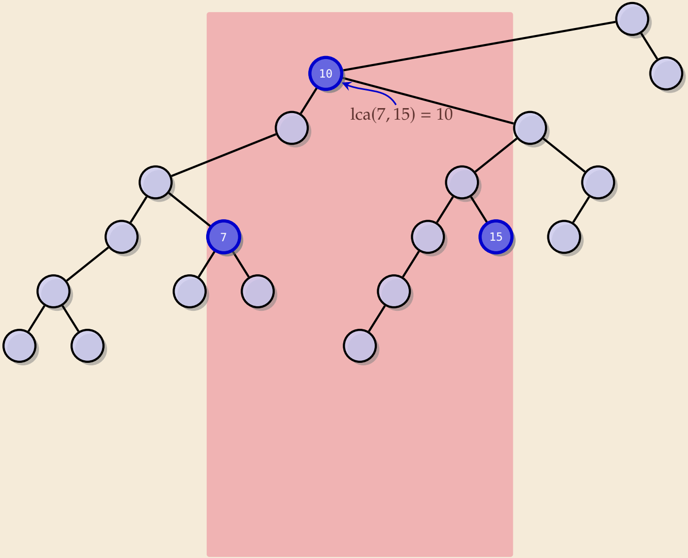
Range-maximum queries



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	6	4	7	10	5	6	3	11	14	2	3	6	10	9	13	4	6	16	10

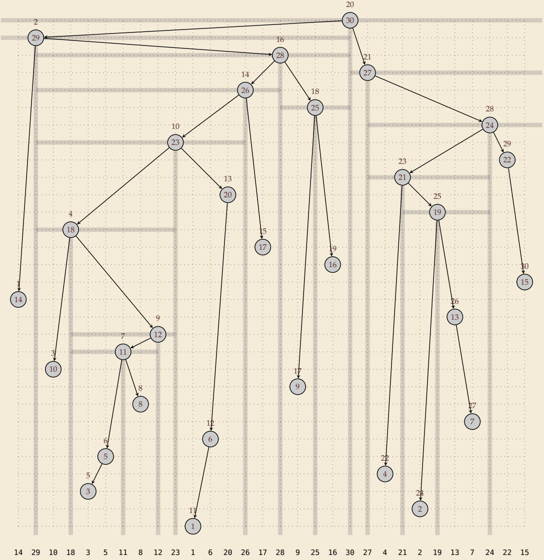
- **Range-max queries** on array A:
  $$\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k] = \textit{index} \text{ of max}$$
- **Task:** Preprocess A, then answer RMQs fast ideally constant time!

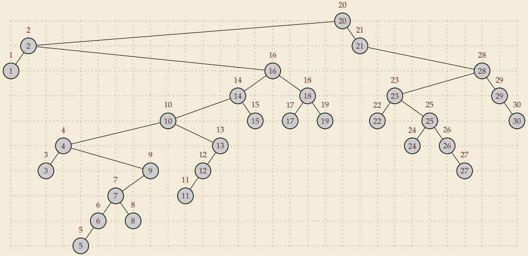
Range-maximum queries



- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i,j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A, then answer RMQs fast ideally constant time!
- **Cartesian tree:** (cf. *treap*) construct binary tree by sweeping line down
- $\mathrm{rmq}(i,j)$ = inorder of **lowest common ancestor** (LCA) of ith and jth node in inorder

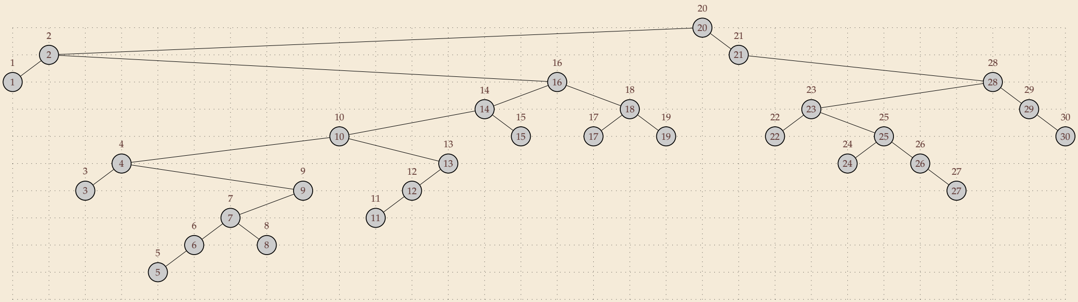
Cartesian Tree – Example





Counting binary trees

- all RMQ answers are determined by Cartesian tree
- How many different Cartesian trees are there for $A[0..n)$?
  - known result: Catalan numbers $\dfrac{1}{n+1}\dbinom{2n}{n}$
  - easy to see: $\le 2^{2n}$



9.5 “Four Russians” Table

Bootstrapping

- We know a $\langle O(n \log n), O(1) \rangle$ time solution
- If we use that for $m = \Theta(n/\log n)$ elements, $O(m \log m) = O(n)$!

- Break A into blocks of $b = \lceil \frac{1}{4} \lg n \rceil$ numbers
- Create array of block minima $B[0..m]$ for $m = \lceil n/b \rceil = O(n/\log n)$
  - $\rightsquigarrow$ Use sparse tables for B.

Query decomposition

Precomputing intra-block queries

Discussion

- $\langle O(n), O(1) \rangle$ time solution for RMQ

$\rightsquigarrow$ $\langle O(n), O(1) \rangle$ time solution for LCE in strings!

👍 optimal preprocessing and query time!

👎 a bit complicated

Research questions:

- Reduce the space usage
- Avoid access to A at query time