9 Range-Minimum Queries

04 May 2021

Sebastian Wild

version 2021-05-04 10:45

Outline

9 Range-Minimum Queries

9.1 Introduction

Range-minimum queries (RMQ)

- **Given:** Static array $A[0..n)$ of numbers (any ordered objects)
  array/numbers don't change
- **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

Clicker Question

Given the array from the slides, what is $RMQ_A(1,6)$ = 1

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

sli.do/comp526

Click on “Polls” tab

Rules of the Game

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

- Two main quantities of interest:

  $\rightsquigarrow$ space usage $\leq P(n)$

  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

Clicker Question

What do you think, what running times can we achieve? For a $\langle P(n), Q(n) \rangle$ time solution, enter “`<P(n),Q(n)>`”.

`sli.do/comp526`

Click on “Polls” tab

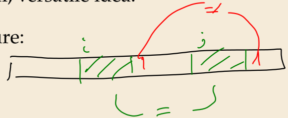
9.2 RMQ, LCP, LCE, LCA — WTF?

Recall Unit 6

Application 4: Longest Common Extensions

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

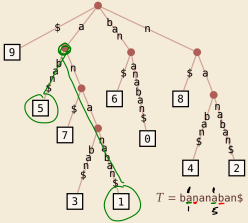
  The *longest common extension (LCE)* data structure:
  - **Given:** String $T[0..n-1]$
  - **Goal:** Answer LCE queries, i. e.,
    given positions i, j in T,
    how far can we read the same text from there?
    formally: $\mathrm{LCE}(i,j) = \max\{\ell : T[i..i+\ell) = T[j..j+\ell)\}$



$\rightsquigarrow$ use suffix tree of T!

longest common prefix of ith and jth suffix

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



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

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.

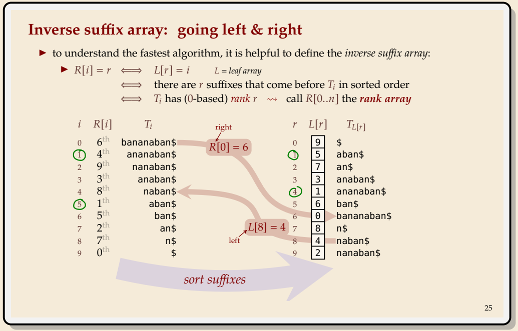
$\rightsquigarrow$ After linear preprocessing (time & space), we can find LCEs in $O(1)$ time.

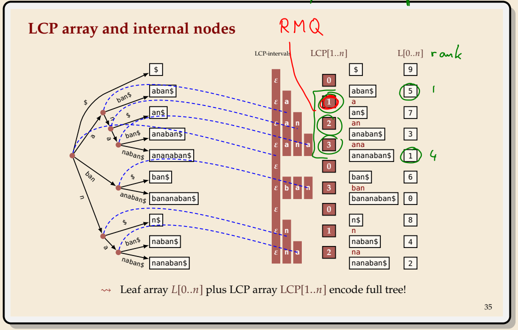
Finally: Longest common extensions

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

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

1 5
bananaban $





RMQ Implications for LCE

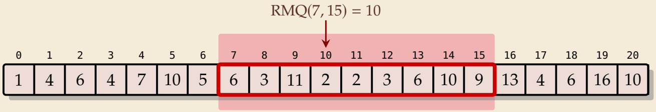
- Recall: Can compute (inverse) suffix array and LCP array in $O(n)$ time

$\rightsquigarrow$ 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

$\Rightarrow$ really want $\langle O(n), O(1)\rangle$ solution
(best possible)

9.3 Sparse Tables

Trivial Solutions



- Two easy solutions show extreme ends of scale:

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

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

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

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

Trivial Solutions

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:

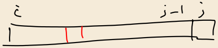
1. **Scan on demand**
   - no preprocessing at all
   - answer $\mathrm{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min

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

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 $\rightsquigarrow$ $\langle O(n^2), O(1) \rangle$



RMQ(i,j) = RMQ(i,j-1)
or j