SCT Lecture: Fischer-Heun RMQ

Stephen Huan

4/2/20

(coincidentally, the day I was supposed to go…)

Meant to be Watched with the “Abridged” Lecture!

- https://github.com/stephen-huan/sct-lectures/blob/master/abridged_rmq/lecture.pdf
- Alternatively… https://github.com/stephen-huan/sct-lectures/blob/master/rmq/lecture.pdf

Definition of Range Minimum Query (RMQ)

- Given an array A with length n and two indexes i, j | i ≤ j, find the smallest element between i and j (inclusive on both ends).
- <f, g> runtime notation

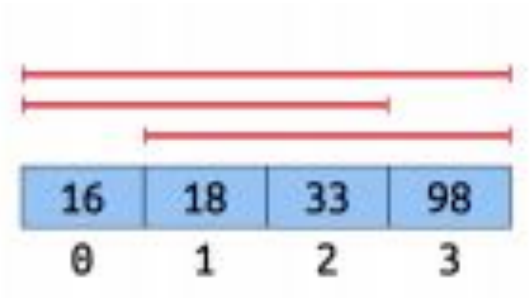
Algorithms Not Discussed

- Fenwick Trees (BITS)
- Segment Trees

Extended Example

- A = [5, 3, 4, 1, 2]

DP Solution



- n^2 possible queries
- A = [5, 3, 4, 1, 2]
    - Precompute them all!

Length of range

Starting index

	1	2	3	4	5
0	5	3	3	1	1
1	3	3	1	1	
2	4	1	1		
3	1	1			
4	2				

Sparse Tables

- Ok, we don't *actually* need to precon
- A = [5, 3, 4, 1, 2]
    - Can answer a general range based off **tw**

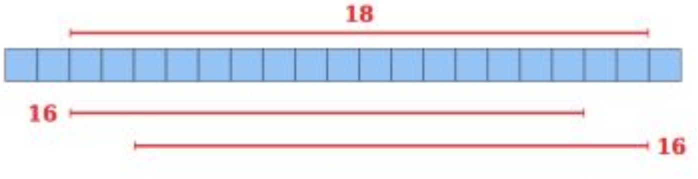


Figure 2: Breaking up a range into powers of two

Length of range

Starting index

	1	2	4
0	5	3	1
1	3	3	1
2	4	1	
3	1	1	
4	2		

Block decomposition

- Break up the array into blocks
- “Square root decomposition” is a special case

Case #2

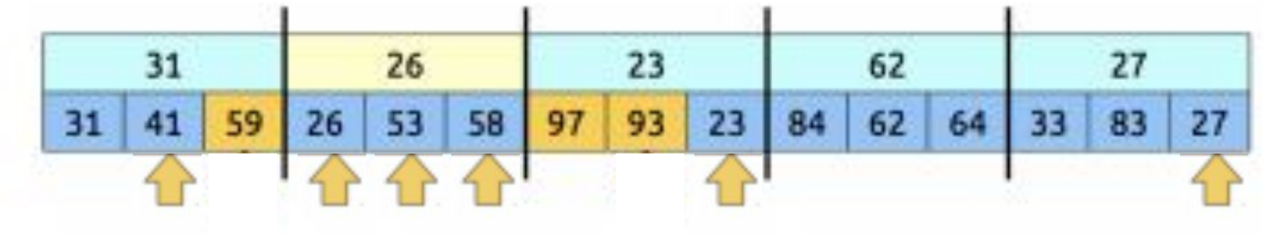


Figure 3: Array with block size $b = 3$

Hybrid Structures

- Use different algorithms for the “top” and “bottom” blocks

Figure 4: Top level view of the overall RMQ structure

Building up to Fischer-Heun: Block Types

- Some blocks aren’t equal, but are “similar”
  - Able to reuse block-level structures

B1 = [1, 3, 2, 4]
B2 = [10, 30, 20, 40]
B1 ~ B2

Detecting Block Types: Cartesian Trees

- Same block type if and only if isomorphic Cartesian trees
    - Implicitly maintain the “morphism” of each Cartesian tree
- Cartesian Tree:
    - Binary tree
    - Inorder traversal yields the original array
    - Min heap

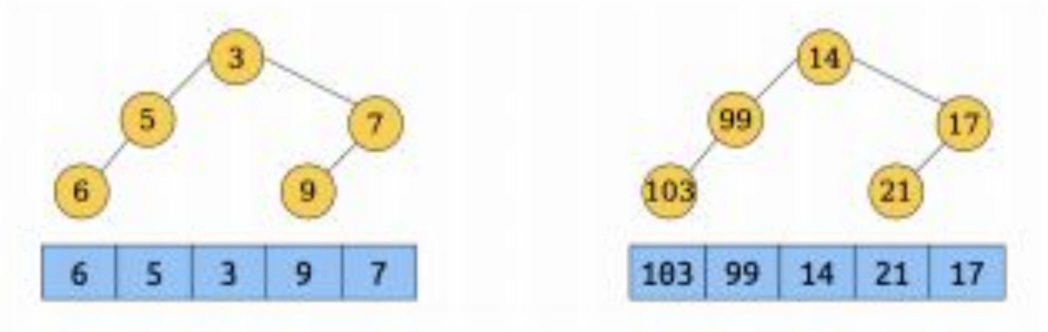


Figure 5: Cartesian trees

Building Cartesian Trees

- Keep a stack of "active" nodes.
- To insert a new node:
  - While the stack is not empty and the top node has a value greater than the new node:
    - Pop nodes off the stack
  - Make the parent of the new node the top of the stack, or null if the stack is empty (the new node is now the root).
  - Make the left child of the new node the last node that was popped off the stack, null if nothing was popped off.
  - Add the new node to the stack.

This algorithm runs in $\Theta(n)$ and does at most $2n$ operations.

Runtime Analysis

- sparse table on the summary array with full precomputation on each block

$$O(n + \frac{n}{b} \log n + (\#\text{ of distinct blocks})b^2)$$

$$O(n + \frac{n}{b} \log n + 4^b b^2)$$

$$O(n + n + n^k (k \log_4 n)^2)$$

$0 < k < 1$
-> $k = \frac{1}{2}$ (why not?)

Fischer-Heun Structure, in Summary

- Set block size to a multiple of the log of the size of the array (usually k = ½)
- Split the array into those blocks and calculate the minimum in each block
- Build a sparse table on the reduced array
- Build DP tables on each block, re-using a structure if it has the same Cartesian tree number
- Answer queries using the usual hybrid approach

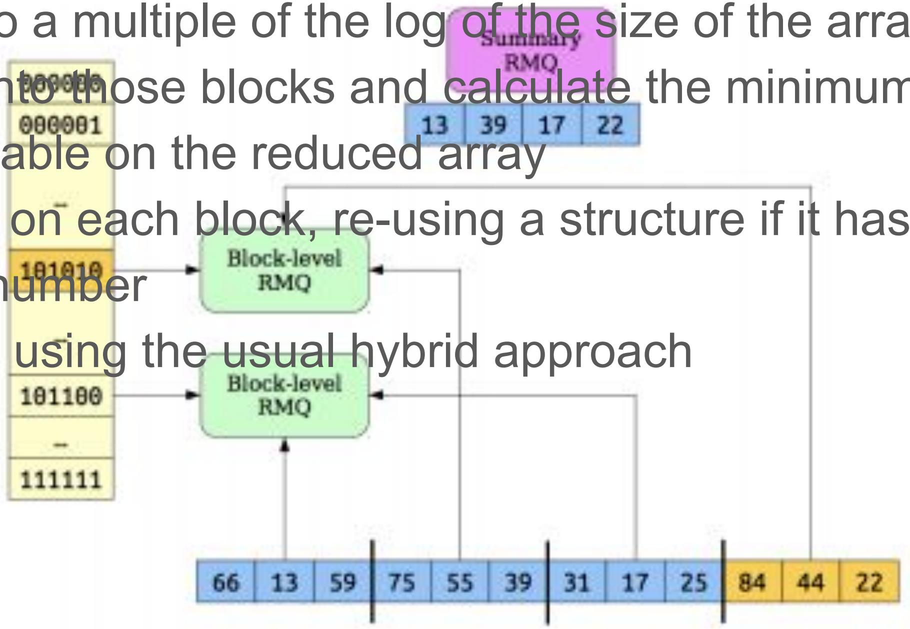


Figure 6: The final RMQ structure

Sample Problem #1: SPOJ RMQSQ

- Direct application of RMQ

#2: USACO Max Flow, or any LCA problem

- Not a max flow problem
- Tree -> array (Eulerian tour)
  - RMQ on this array

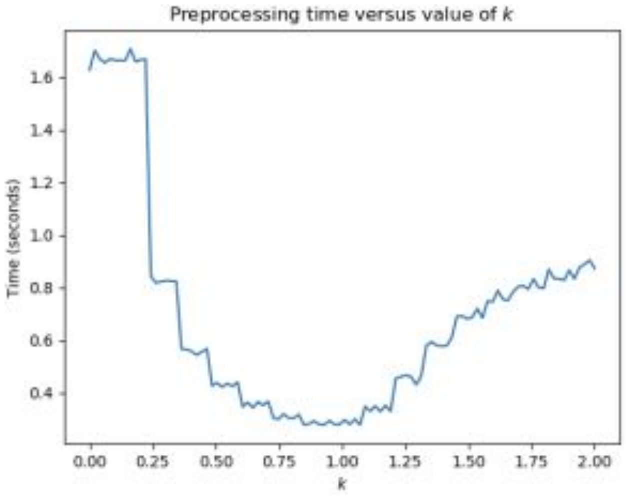
#3: SPOJ BEADS, SPOJ LPS, Leetcode LPS, Leetcode palindromic-substrings, etc.

- Define Longest Common Prefix (LCP)
  - LCP(“abcd”, “abef”) = 2
- Suffix array and LCP Array
- LCP value between two nonadjacent suffixes is the range minimum
  - RMQ!

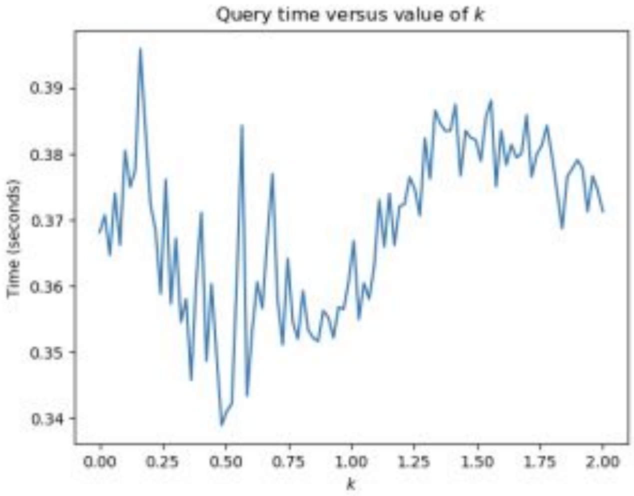
#4: Sliding window/monotonic queue

- Just do RMQ(i, j) where i and j are the start and end positions of the window/queue

Side Note: picking *k*



(a) Preprocessing time



(b) Query time

k = 1 wins! (0.8 if you want to be a theoretical purist)

Sources

All images (except for the ones on the “picking k” slide) and ideas come from:

1. https://web.stanford.edu/class/cs166/lectures/00/Slides00.pdf
2. https://web.stanford.edu/class/cs166/lectures/01/Slides01.pdf

SCT lectures mentioned can be found on the last page of the full RMQ lecture

(not the abridged version)

5 Past Lectures

1. BITs
   (a) “Binary Indexed Trees” (Patrick Zhang, 2019)
   (b) “Binary Indexed Trees” (Daniel Wisdom, 2018)
   (c) “Binary Indexed Trees” (Justin Zhang, 2017)
   (d) “Binary Indexed Trees” (Nick Handay and Ryan Jian, 2012)
2. Segment trees
   (a) “Segment Trees” (Richard Zhan, 2019)
   (b) [illegible] Segment Trees” (George Tang, 2018)
   (c) “Segment Trees” (Justin Zhang, 2017)
   (d) (Broken) “More Segment Trees” (Kevin Geng, 2017)
   (e) (Broken) “Segment Trees” (Charles Zhao, 2016)
   (f) (Broken) “$\sqrt{n}$ Bucketing and Segment Tree” (Samuel Hsiang, 2015)
   (g) (Unavailable) “Segment Tree (and its variants)” (Wassim Omais and Shwetank Patel, 2016)
3. Lowest Common Ancestor
   (a) “Lowest Common Ancestor” (Richard Zhan, 2019)
   (b) “Lowest Common Ancestor” (Daniel Wisdom, 2017)
   (c) “Lowest Common Ancestor” [most similar to this lecture] (Matthew Savage, 2015)
   (d) (Broken) “LCA and 2^n Jump Pointers” (Larry Wang, 2016)
4. Longest Common Prefix
   (a) “Suffix Arrays and Longest Common Prefix” (Daniel Wisdom, 2019)
   (b) Constructing LCP array
5. “January Contest Review ($\sqrt{N}$ decomposition”) (Justin Zhang and Daniel Wisdom, 2018)
6. “Simple Range Queries” (Justin Zhang, 2017)
7. “Advanced Data Structures” [for RMQ] (Alex Chen, 2011)