Individual Assessment 1: RMQ

This is an individual assessment, and, as the name suggests, must be completed individually. Specifically, you're not allowed to work with a partner, and you should not discuss these problems with other students in CS166. However, the course staff are happy to answer clarifying questions on EdStem (if you do, please post the question privately) or in our office hours.

Due Tuesday, April 12th at the start of lecture

Problem One: Area Minimum Queries

In what follows, if A is a 2D array, we'll denote by A[*i*, *j*] the entry at row *i*, column *j*, zero-indexed. When drawing out a grid, we'll assume that (0, 0) represents the upper-left corner of the grid, that (1, 0) represents the entry in the leftmost column of the row just beneath the top row, and that (0, 1) represents the entry in the top row, just to the right of the upper-left corner.

This problem concerns a two-dimensional variant of RMQ called the ***area minimum query*** problem, or ***AMQ***. In AMQ, you are given a fixed, two-dimensional array of values and will have some amount of time to preprocess that array. You'll then be asked to answer queries of the form "what is the smallest number contained in the rectangular region with upper-left corner (*i*, *j*) and lower-right corner (*k*, *l*), inclusive?" Mathematically, we'll define $AMQ_A((i, j), (k, l))$ to be $\min_{i \le s \le k,\, j \le t \le l} A[s, t]$. For example, consider the following array:

31	41	59	26	53	58	97
93	23	84	64	33	83	27
95	2	88	41	97	16	93
99	37	51	5	82	9	74
94	45	92	30	78	16	40
62	86	20	89	98	62	80

Here, A[0, 0] is the upper-left corner, and A[5, 6] is the lower-right corner. In this setting:

- $AMQ_A((0, 0), (5, 6)) = 2$
- $AMQ_A((0, 0), (0, 6)) = 26$
- $AMQ_A((2, 2), (3, 3)) = 5$

For the purposes of this problem, let *m* denote the number of rows in A and *n* the number of columns.

i. Design and describe an $\langle O(mn), O(\min\{m, n\})\rangle$-time data structure for AMQ.

ii. Design and describe an $\langle O(mn \log m \log n), O(1)\rangle$-time data structure for AMQ.

As a hint, think about what these constraints say if you're looking at, say, a $1 \times n$ array, which should essentially reduce to a regular RMQ structure.

When writing up your answers, please follow the general advice from our "Assignment Policies" handout. Begin with a high-level overview of the approach, then drop down into lower levels of detail and explain how each operation works, concluding with a runtime analysis.

Fun fact: You can improve these bounds all the way down to $\langle O(mn), O(1)\rangle$ using a Four Russians speedup and some very clever auxiliary data structures. This might make for a fun research project topic if you've liked our discussion of RMQ so far!

Problem Two: On Constant Factors

As a refresher, here's how the preprocessing step for the Fischer-Heun RMQ structure works:

- Split the input array into blocks of size $b = \frac{1}{2} \log_4 n$.
- Construct an array of the block minima and build a sparse table RMQ on the minima.
- Construct an array of size 4^b storing pointers to RMQ structures, initially all null.
- For each block in the input array, compute its Cartesian tree number. If the array entry for that number is null, create a new $\langle O(n^2), O(1) \rangle$ precompute-all RMQ structure and store it in the array at that point. At this point, the array slot at that index is definitely not null, so store a pointer from the current block to that RMQ structure.

You may have noticed that we specifically picked $b = \frac{1}{2} \log_4 n$ as its block size. Why is that ½ there? Why is this a base-4 logarithm? This question explores the answer to that question.

i. Redo the analysis of the Fischer-Heun preprocessing time we did in class (the one starting on Slide 247 of the main slide deck) under the assumption that $b = \log_2 n$. What big-O runtime bound do you get?

   Just to make sure we're clear here, the only change you're making to the Fischer-Heun data structure is dropping the leading coefficient from b. The rest of the approach remains the same.

The bound you came up with in part (i) is not "tight," in the sense that if $p(n)$ is the true worst-case preprocessing cost for the modified Fischer-Heun structure, then $p(n) = o(f(n))$, where $f(n)$ is the bound you came-up with in part (i) of this problem and o denotes little-o notation.

ii. Give a tight bound on the preprocessing cost of the modified Fischer-Heun structure. Some hints on this problem:

   - Feel free to use the fact that the number of distinct block types of size b is given by C_b, where C_b denotes the bth Catalan number.
   - Consider the relationship between how many blocks an input array of length n consists of and the number of possible blocks of size b. Which is bigger? By how much?
   - Your analysis will have to deviate significantly from the one we did in class, since as mentioned above, that analysis will overestimate the total work. Go back to basics, accounting for the cost of each step of the preprocessing algorithm in the worst case.
   - The analysis you did in part (i) not only overestimates the amount of work done, but *significantly* overestimates the amount of work done. You should see a major reduction in the bound, not just, say, shaving off a factor of log n or log log n or something like that.

Generally speaking, it's good to interrogate constant factors and other seemingly arbitrary design decisions in a data structure. Sometimes, those choices really are arbitrary and can be tuned for performance reasons. Other times, those choices are there for a specific reason, and seeing why helps you better understand why things work the way they do.