Week 2: Range Queries, Sweeps and Geometric Transformations

Authors: Allen Pei and Ava Pun
Edited by: Richard Peng, Neal Wu
Incorporated comments from:

In many problems, you will have to do range queries: Given an array, update the elements in range [l, r], or find some property of the elements in range [l, r]. Even when you aren't explicitly asked to do these queries, they can be useful components of your solution. For example, in some DP problems, you can turn a factor of N into $\log N$ by using efficient range queries (e.g. CCO '21 P5 - Bread First Search). Variants of range queries include 2D range queries, queries on trees, etc.

Common data structures to support range queries include binary indexed trees, segment trees, and (balanced) binary search trees. Segment trees, being more versatile than BITs (binary index trees) and easier to code than BBSTs (balanced binary search trees), are widely used.

Here we will discuss some stranger uses of range queries, and tricks that you can use when faced with these kinds of problems.

(if this picture does not cause alarm, please solve more ra(n)ge query problems)

Range GCD, Range add

Problem: Write a data structure to support the following queries on array $[a_1, \ldots, a_N]$:

- Add x to range [l, r]
- Find GCD of elements in range [l, r]

Solution: Notice that $\gcd(a_l, a_{l+1}, \ldots, a_r) = \gcd(a_l, a_{l+1} - a_l, a_{l+2} - a_{l+1}, \ldots, a_r - a_{r-1})$. So we can keep 2 segment trees:

- Tree 1 supports range add updates and point queries over $[a_1, \ldots, a_N]$
- Tree 2 supports point updates and range GCDs over $[a_2 - a_1, a_3 - a_2, \ldots, a_N - a_{N-1}]$

For an add query, we can easily update tree 1, and we only need to do 2 point updates in tree 2. And for a GCD query, we can use tree 1 to find a_l and tree 2 to find $\gcd(a_{l+1} - a_l, a_{l+2} - a_{l+1}, \ldots, a_r - a_{r-1})$, then combine the answers.

Variant of this trick: https://codeforces.com/problemset/problem/587/E

Record the queries

This is more of just a nice trick to notice. In some problems with "queries" the naive bound (# of queries) x (worst-case time per query) is too large. But if you memoize the queries, it actually runs fast enough. For example: https://open.kattis.com/problems/kamioni. I solved it in time ~O(K * sqrt K) where K = max(N, M) simply by processing each query in ~O(a) time where a is the length of the shorter of the two sequences for the query (this takes some preprocessing). Naive analysis only shows this runs in ~O(K^2) time, but if you memoize the queries to ensure you never compute the answer twice for the same query pair, you can show it runs in ~O(K * sqrt K) time. Here, the ~O() is ignoring a single logarithmic factor.

https://dmoj.ca/problem/ioi09p6

What's often called Mo's is another version of this

https://danang19.kattis.com/problems/easyquery (on one of the practices earlier this year) is partial query state memorization plus a (persistent?) segtree

Sweep

Oftentimes, a 2D query problem can be turned into a 1D problem by sorting the queries and doing a sweep down the y-direction. That way, you only have to maintain a data structure in the x-direction. Example: DMOPC '18 Contest 6 P5 - Quadrat Sampling.

WF12I https://open.kattis.com/problems/safebet (check if a pair of line segments intersect) is the classical ‘reduce dimensionality by 1’ problem.

More examples of sweep problems:

https://dmoj.ca/problem/noi10p2

https://dmoj.ca/problem/noi15p6 (we are in the midst of configuring this problem to throw at high schoolers preparing for the NOI, let us know when you submitted so we can check our checker)

https://open.kattis.com/problems/oil2
https://open.kattis.com/problems/saskatchewan

Transformations

With some geometry problems, you can “change the basis” to make things easier to work with.

For example, Exhaustive Experiment involves triangular sectors of the plane. But if you scale the triangles horizontally by a factor of 2, then rotate them by 45 degrees, they become axis-aligned rectangles, which you can easily sweep over.

Another cool trick is that if you rotate a pair of points (x_1, y_1), (x_2, y_2) by 45 degrees to get (x_1', y_1'), (x_2', y_2'), the Manhattan distance between the original points can be written two ways:

$$|x_1 - x_2| + |y_1 - y_2| = \max(|x_1' - x_2'|, |y_1' - y_2'|)$$

This new form can be easier to work with. For example, if you want to maximize the Manhattan distance between two points, you can separately try maximizing the vertical and horizontal differences between the rotated points.

WF11E
IOI07 Pairs: https://dmoj.ca/problem/ioi07p5
IOI16 Shortcut: https://dmoj.ca/problem/ioi16p3
https://codeforces.com/contest/1534/problem/G