Data Structures II

Junhua Chen 13/12/21

Table of contents

Abstract:
We discuss 3 interesting and powerful ways to utilize segment trees

Section I: Pushup functions

(1) Thinking outside box with merge functions

(2) The bracket matching segment tree and applications

- ***(i) max subarray sum***
- ***(ii) Problem: Memes***

(3) Dynamizing DP with segment trees

Section II: The segment tree as a powerful multiset

(1) Common applications: Prefix Sum Maximum, kth element

(2) Example application to a greedy problem

Section III: Sets and priority queues in segment trees

(1) Lowering standards: Lazy priority queues in segment trees

(2) Sets: solving 2D problems via problem <<Charlie the Wondermondarian>>

Revisiting what segment tree is

- A segment tree (well, non lazy ones anyway) can be described by
  (i) What each node stores
  (ii) How the nodes are merged (the ‘pushup’ function)
- This really does give you a lot more room for imagination than you expect
  - Example: (integer, max) segtree is just RMQ

A data structure problem to press my case

Maintain:

An graph with $N \leq 100$ nodes and an set of $M \leq 10^5$ edges each with a unique label $1 \ldots M$ (i.e an array of edges)

Query ($Q \leq 5 \times 10^4$):

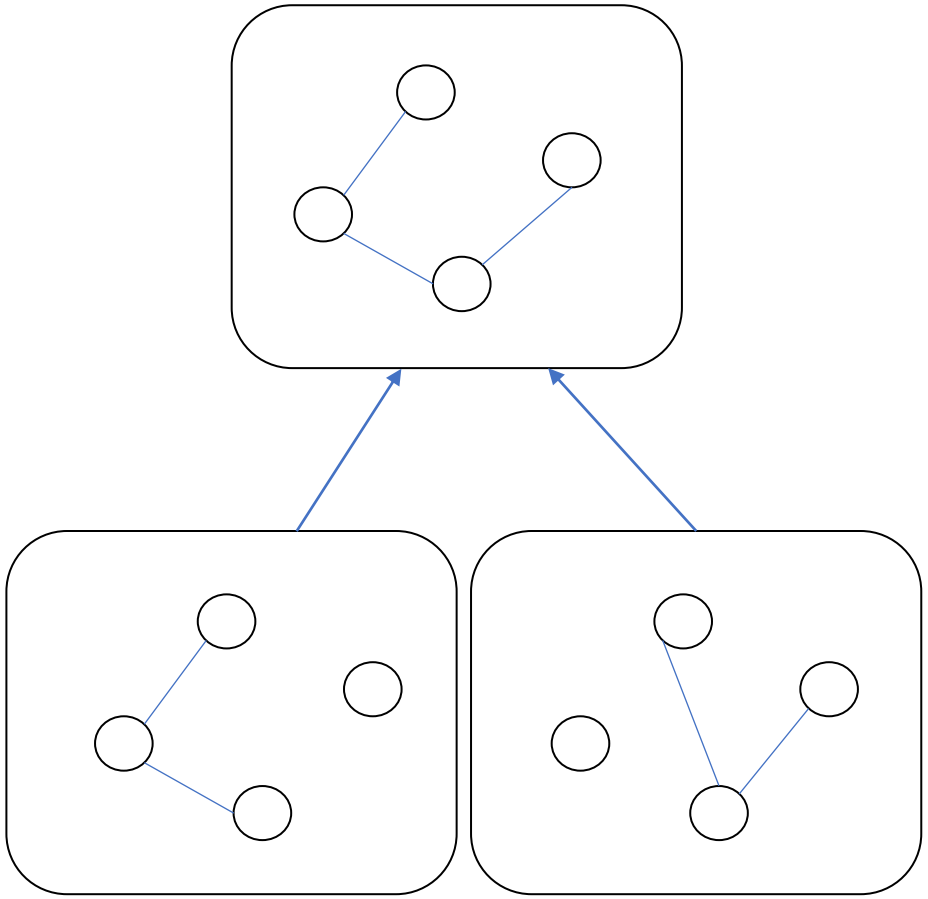
Given l, r how many connected components are there if we only keep edges with labels $l \ldots r$

Cool solution

- Segment tree node $[l, r]$ stores spanning tree made by edges $l \ldots r$
- Merge in $O(N)$ by combining spanning trees and rerunning dfs

- $O(QNlogN + MN)$ complexity (note initialization segment tree consists of M−1 merges at linear time per merge)

- *Key point:* Segment trees are much more than just range sum / range max devices!
- *Optional:* Try to find $O(QlogN + MN)$ solution

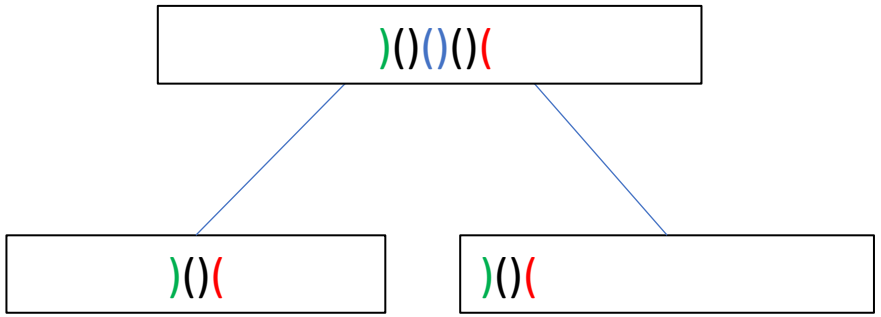


The bracket matching segment tree

- Motivating problem: https://codeforces.com/problemset/problem/380/C

How to design our segment tree

- What does our node $[l, r]$ store?
  - Max balanced subsequence in range
  - Number of unused) brackets
  - Number of unused (brackets



- Pushup function:
  - Match greedily (intuitively is optimal)

- **This idea of 'divide and conquer' and greedy pairing merge is very useful and common (we're basically embedding D&C solution into a segment tree)**

Implementation: Observe most of code is generic fluff!

- https://codeforces.com/contest/380/submission/138484514

Maximum subarray sum with updates

- Maximum subarray sum but with point updates & range queries

- Keep:
  - Range max subarray sum
  - Range max prefix sum
  - Range max suffix sum
  - Range sum (for calculating prefix & suffix max)

- Use observation that splitting the array in the middle either cuts the max subarray in two or doesn't
  - $ans_{l \cup r} = \max(ans_l, ans_r, ans_{split})$
  - D&C and 'matching left and right' again!

Memes

- https://orac2.info/problem/aiio14memes/
- WLOG critical sequence is $1 \ldots M$
- TL DR: Given text length N solve the problems, point text updates
  - Find number of times the string $1 \ldots M$ appear in text
  - Find max number of times the string $1 \ldots M$ is repeated adjacently
- Clearly it is a segment tree problem and involves a carefully designed pushup function to compute the answer
- Lets discuss what we need to store (once we figure this out the pushup function becomes easy−ish)

Solution: What does a segtree node store

Part I: Number of Runs

- Number of runs in range
- Is the range covered by node a part of a single run?
- For prefix & suffix whether its prefix/suffix of run

Part II: Longest number of consecutive repetitions

- Longest consecutive repetition in range
- Longest prefix & suffix repetition
- Is whole range a repetition

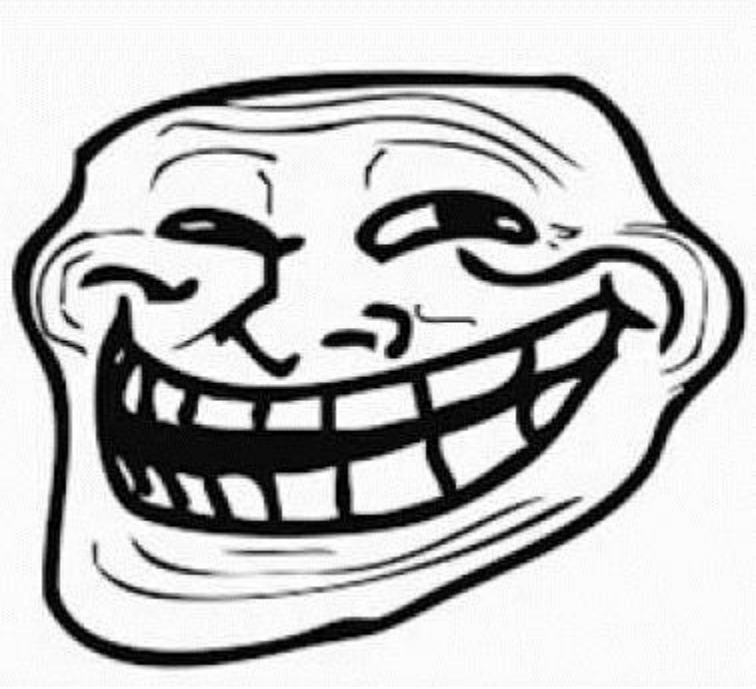
Mental relaxation break

Here are two easy DP problems: have 30 seconds to solve them

- Given an array of N integers we select some of these integers. Find the maximum sum of selected integers if no two adjacent entries can be selected (max nonadjacent subarray sum)
- Given a 5 by N grid with some squares blocked out find the number of paths from (1,1) to (5,N) only going down and to the right (say modulo 1e9+7) *[Extension: What if we can go in any 4 directions, although each square can be traversed at most once]*
- $1 \leq N \leq 10^5$

NOW DO IT WITH UPDATES!!!!!

- How do we slap the DP in the segtree?



I’ll leave most of the details to you guys (theres not really a problem for this in set)

- Just store the left and right frontier states of each range and merge in the ‘reasonable’ way

- In second case $ST_{[l,r]}$ stores for each $1 \leq i \leq j \leq 5$ $ways[i][j] = \#ways\ to\ go\ from\ (l,i) \rightarrow (r,j)$
  - $ways_{[l,r]}[i,j] = \sum_{k=i}^{j} ways_{[l,mid]}[i][k] ways_{[mid+1,r]}\ [k][j]$
  - Bit like matrix multiplication if you know that(don’t worry if you don’t)
  - $O(5^3 logN)$ per update

General tips

- Writing down what your segment tree needs to do is very useful to solving problems
  - Be better able to understand what your tree needs to maintain
  - And thus better understand what your pushup needs to do
- Abstracting the segtree's operations as a black box also helps
  - allows you to focus on the problem solving elements at hand
  - Be able to deal with purely 'segtree design' elements at the end (use intuition to guide whats possible)
  - As soon as you get the segtree, point updates and range queries trivial

- *one should not try to fit a problem into the mould of a common segment tree, but should fit a segment tree if possible into the problem*. – A wise person

Section I: Pushup functions

(1) Thinking outside box with merge functions

(2) The bracket matching segment tree and applications

(i) max subarray sum

(ii) Problem: Memes

(3) Dynamizing DP with segment trees

Section II: The segment tree as a powerful multiset

(1) Common applications: Prefix Sum Maximum, kth element

(2) Example application to a greedy problem

Section III: Sets and priority queues in segment trees

(1) Lowering standards: Lazy priority queues in segment trees

(2) Sets: solving 2D problems via problem <<Charlie the Wondermondarian>>

Quick exercise: How do we solve the following task quickly (1 min)

- Maintain set of integers with insertions, deletions & queries:
  - “find kth element” for different k
  - “how many numbers between l and r”
  - “sum all elements less than k”

Solution

- You might think that we need to use SBBST for this but not really

- Just maintain lazy create segment tree on frequency array
  - Now everything is easy! Can we see how we design the pushup function?
  - Example: set is {1,1,2,2,2,3,3,4}
  - This is really useful for many problem situations and makes life simple

Value	1	2	3	4
Frequency	2	3	2	1

Example Problem: Pancakes

- https://acio-olympiad.github.io/2021/pancakes.pdf

- TL DR: N types of pancakes, p_i of type i. In move you can eat K pancakes of different types.
  - Query: How many pancakes can you eat?
  - Update: The value of p_i for some i

  - $1 \leq N, K, Q \leq 10^5$

Observation

- One can make x moves iff $\sum_{i=1}^{N} \min(p_i, x) \geq Kx$
- The focus of lecture is data structures so we'll skip the proof
  - Ideally try and prove it yourself
  - If your really lazy editorial is at bottom here https://cdn.discordapp.com/attachments/727851161071386644/808985540757946398/acio2020contest1_editorial.pdf

Solution

- Binary search on answer (the previous criteria forms our decision function). However we need to account for updates so the decision function must be executed quickly

Lets practice writing the data structure requirement:

Maintain:

Update:

Query:

Solution

- Binary search on answer (the previous criteria forms our decision function). However we need to account for updates so the decision function must be executed quickly

Lets practice writing the data structure requirement:

Maintain: $P[1 \ ... N]$

Update: $P[i] \coloneqq v_i$ (point update)

Query: $\sum_{i=1}^{N} \min(P[i], x)$ for some x

Okay so how do we do this

- $\sum_{i=1}^{N} \min(P[i], x) = \sum_{P[i]<x} P[i] + (\#P[i] \geq x) \times x$
- Count number of types with more than x pancakes: range sum on frequency array (i.e store frequency sum in range)
- Sum all types with less than x pancakes: weighted range sum on frequency array (i.e store weighted sum in range)

- $O(logMAX)$ per query
- Lets look at implementation

In all …

- $O(\log MAX)$ per evaluation of decision function
- $O(\log MAX)$ calls of decision function for binary search per query
- $\therefore O(\log^2 MAX)$ per query and easy to implement

Section I: Pushup functions

(1) Thinking outside box with merge functions

(2) The bracket matching segment tree and applications

(i) max subarray sum

(ii) Problem: Memes

(3) Dynamizing DP with segment trees

Section II: The segment tree as a powerful multiset

(1) Common applications: Prefix Sum Maximum, kth element

(2) Example application to a greedy problem

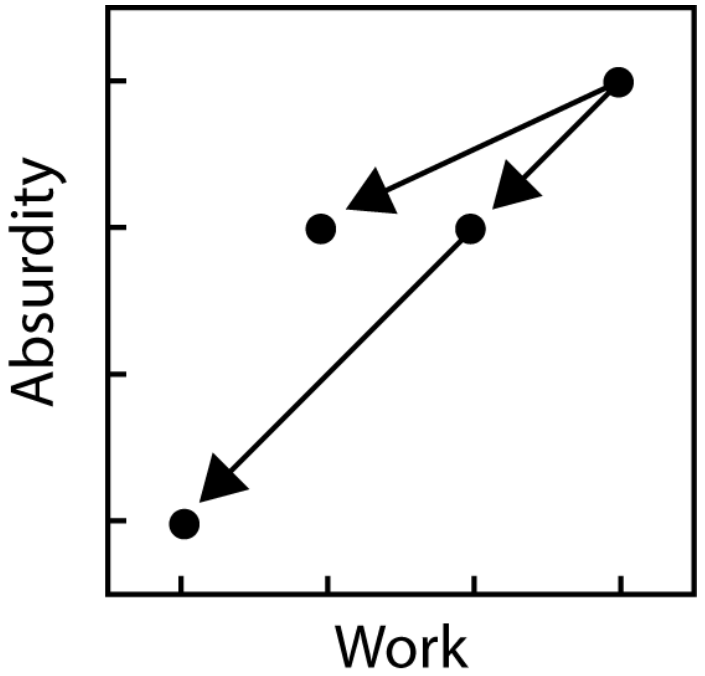
Section III: Sets and priority queues in segment trees

(1) Lowering standards: Lazy priority queues in segment trees

(2) Sets: solving 2D problems via problem <<Charlie the Wondermondarian>>

Lowering Standards

- ~~Me when going to restaurant in England~~
- https://orac2.info/problem/seln15standards/
- TL DR given set of tasks $(w_i, a_i, d_i) \in T$ find longest sequence of distinct tasks $T_1 \ldots T_k \in T$ such that $0 < w(T_{i-1}) - w(T_i) \leq W$ and $0 < a(T_{i-1}) - a(T_i) \leq d(T_{i-1})$ [is this okay?]
  - In words of problem statement T_i is backup to T_{i-1}



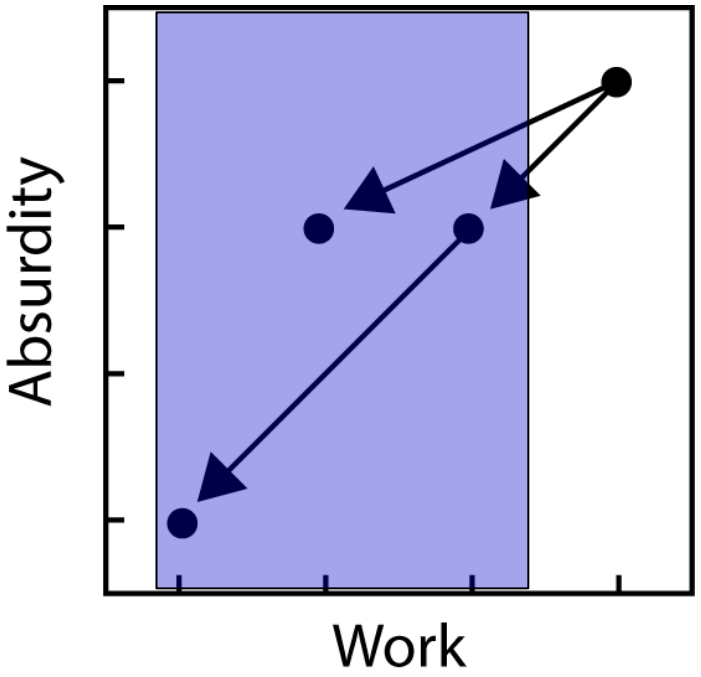
First thoughts

- Very similar to LIS
- Sweep along W (work) column which is well behaved
  - For each task pick optimal predecessor with longest backup chain (LIS style)
  - Maintain data structure that can query for this

- Data structure requirements:
  - **Maintain:** Set of tasks
  - **Query** $(\boldsymbol{w}, \boldsymbol{a_l}, \boldsymbol{a_r})$**:** Find task with $\mathrm{w} - \mathrm{W} \leq w', a_l \leq a' \leq a_r$ and whose chain size (represented by a weight) is maximised
  - **Update:** Insert task

Implementing the data structure

- Ideally if $W = \infty$ our job will be relatively easy, just range query on a segment tree maintaining the best tasks for each a
- But now we need to delete tasks that are too old
- Solution 1 ($O(N \log N)$):
  - Maintain second sweep trailing W behind first sweep and deletes old tasks.
  - We maintain sets for each d_a that store the set of 'active' tasks with that d_a
  - Update segtree leaf values as needed



Solution 2: Segtree of priority queues

- Each node of segment tree maintains priority queue
- The priority queue of $[l, r]$ contains all tasks with $l \leq a \leq r$ and is ranked by the chain length
- **Update:** Insert T into the priority queues of the $O(\log MAX)$ nodes whose range containing $a(T)$
- **Query:** The only issue is deriving the value of a segtree node
  - We pop elements off PQ until we hit an 'active' task
  - Time complexity amortises to $O(N \log N \log MAX)$
  - Slower but quite cool (you've probably used the lazy PQ idea when you code Dijkstra because there's old junk inside the PQ)
  - Lets look at implementation

Segment Tree of Sets – Why is it useful?

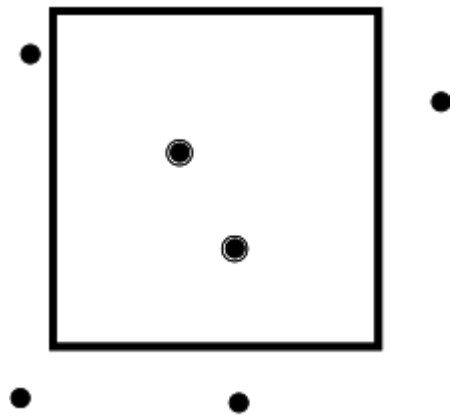
- Do 2D queries on 1 dimensions worth of work (STL does the 2nd)
- Things you can do include (fully dynamically):
  - Find a point/interval in a 2D box or determine none exists [we'll focus here]
  - Find an interval contained wholly within a query interval from a set of them
- It is more convenient than 2D segment tree and faster

A Toy Problem

Maintain: set of points

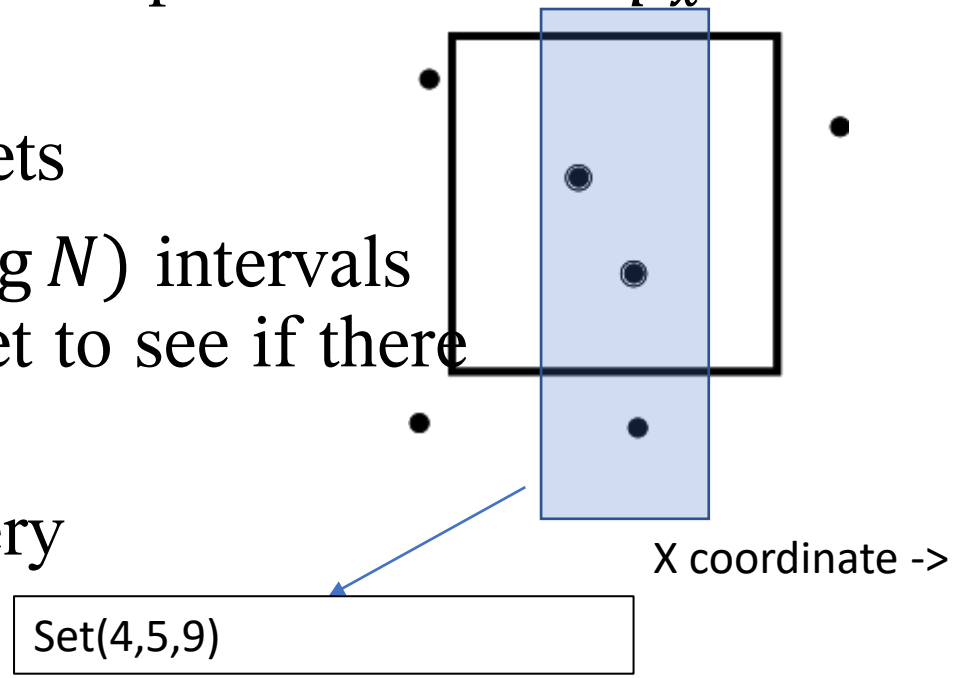
Query: Given a query rectangle is there an point contained within the rectangle

Updates: Insert / Delete Points



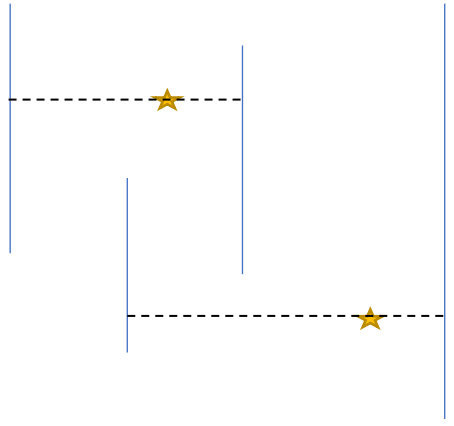
Solution

- Build the segment tree on the X coordinate
- The node $[l, r]$ contains set which keeps all points with $l \leq p_x \leq r$ but now it is keyed by y dimension
- **Update:** Insert point into all relevant sets
- **Query x_1, y_1, x_2, y_2:** For each of $O(\log N)$ intervals covered by $[x_1, x_2]$ we query its set to see if there are any points in $[y_1, y_2]$
- Complexity: $O(\log^2 N)$ update and query



Ways we could have extended & Charlie the Wondermondarian

- **In class Exercise 1:** Extend previous solution to find the point with greatest y coordinate
- **In class Exercise 2:** Rather than points now:
  - We have a set of vertical segments to maintain
  - Our query is a point (x, y): find the segments with minimum x coordinate distance from point and whose y coordinates intersect with y (see diagram)
- Once you read the problem you'll realise exercise 2 effectively solves <<Charlie>>, albeit with difficult implementation
- Lets look at my horrible code



Essential Problems (* = template provided)

Section I	Section II	Section III
Sereja and Brackets *	*Pancakes* *	*Lowering Standards*
Max nonadjacent subarray sum	*Mountain*	
Memes		

Somewhat Harder Problems

Section I	Section II	Section III
Profit Swings (find elegant way!)	Happiness	Charlie the Wondermondarian
Purview		Generating Synergy

GL HF

Very Hard Problems
Skates (POI)
Pyramid Base
AI.io
Election (BkOI 18, oj.uz)
Seats (IOI2018) [oj.uz]
Tenis (COI2020) [oj.uz]

Further Reading

- Mergesort trees (special case of segment tree of sets but shaves off log factor, useful in select situations)
- Segtree Beats: https://usaco.guide/adv/segtree-beats?lang=cpp
- Offline dynamic graph connectivity by divide and conquer
- Persistent segment trees (O(logN) static 2D range queries + range kth element and other good stuff)
- My article (shameless plug) which has an additional section on small merging segment tree: https://anonymous3141.github.io/2021/02/10/Segment-Trees.html

Thank you for your attention!