2014 UESTC Training for Data Structures

Problem B

- Segment tree, Binary Indexed Tree

Just the ordinary operations of the data structure , no more further skills.

Problem C

- Segment Tree

Segment Tree , which should maintain functions that can modify and query intervals. So , `lazy-tag` should be used to optimize the data structure.

Problem D

- Segment Tree + Discretization + Point Division

At first , in order to transfer this problem solving by Segment Tree, we are suggested to turn the frame into a scanning line. To achieve this, we are need to divide a single point into four points , whose `x` coordinate should be `x` and `x + W` , `y` coordinate should be `y` and `y + H`. By doing this, we successfully transfer this problem into `Scanning line problem`. However, pay attention to the data range , a bold Segment Tree is not acceptable as result of the memory limits. All we should do at first is to discretize the data.

Problem E

- Binary Indexed Tree

The problem can be transformed into “Count the numbers before and after which are smaller than a number”. `BIT[num]` indicates the numbers smaller than or equal to `num`, using Binary Indexed Tree to maintain it. Go through the numbers from head to tail and then from tail to head, thus we can store the value of each number in arrays `F[]` and `G[]`. In the end, all you need is to multipy `F[]` with `G[]`.

Problem F

- Binary Indexed Tree

Glancing at it, you may think this problem should be solved by Planar Segment Tree. However , the ranges of the data deny it. :(
We should adapt some tricks solving this problem. Because all the query are like this form `(0 , 0 , x , y)`. So we should `sort` the coordinates at first , `y` as the first key word and `x` as the second. After this , we can use `BIT[num]` to check how many numbers *currently* are smaller than or equal to `num`. Obviously, here `num` should be the `x` coordinate.

> We should add all the coordinates by **1** in order to maintain the `BIT[]` easily.
> i.e.
> `arr[i].x ++;`
> `arr[i].y ++;`

Problem G

- Segment Tree

This problem help us understand Segment Tree profoundly. Because we need to maintain four fields.

- seg_sum(the interval sum)
- seg_max(the max consecutive interval sum)
- seg_max_left(the max consecutive interval sum starts from left side)
- seg_max_right(the max consecutive interval sum starts from right side)

So, How to maintain these fields, it is your task.

Problem H

- Heap

Here we can maintain 2 heaps to solve this problem. We create a `MaxHeap` and a `MinHeap`. Each time we insert a number to the array, we either insert it into the `MaxHeap` or the `MinHeap`. So , the median number will only be either the top of `MaxHeap` or the top of the `MinHeap`. We should also adapt some stratigies to maintain the two heaps in order that the sizes of them are equal or the delta is only **1**.

We should disscuss the following situations:

```
    num < maxHeapTop
        maxHeap.size() <= minHeap.size()
        maxHeap.size() == minHeap.size() + 1
    maxHeapTop <= num <= minHeapTop
        maxHeap.size() <= minHeap.size()
        maxHeap.size() == minHeap.size() + 1
    num > minHeapTop
        minHeap.size() <= maxHeap.size()
        minHeap.size() == maxHeap.size() + 1
```

Problem J

- Trie

Just the ordinary implementations of Trie , no more further skills.

Problem K

- Stack

Just note that each number vary from another. So just implement a stack simply. That's OK.

If you have some questions or find something wrong in my solutions, just contact me in by the following two methods:

- Telephone: (+86) 135 4139 1821
- E-mail: Allen.3.ysj@gmail.com

The source code will be staged on my github:

URL: https://github.com/Allen3/CDOJ