Line Sweep Algorithm Concepts & Qns

codestorywithmik

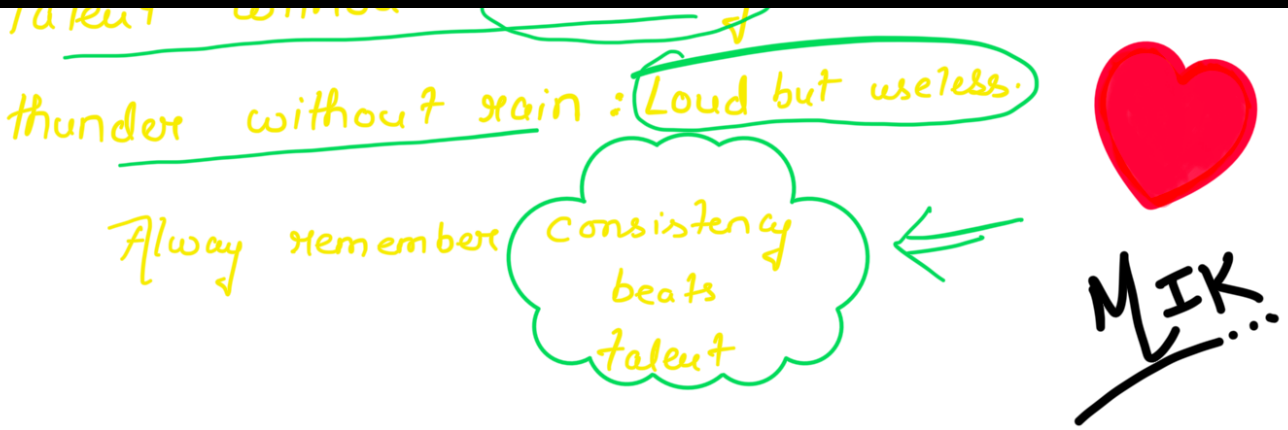
CSwithMIK

codestorywithMIK

Video - 8

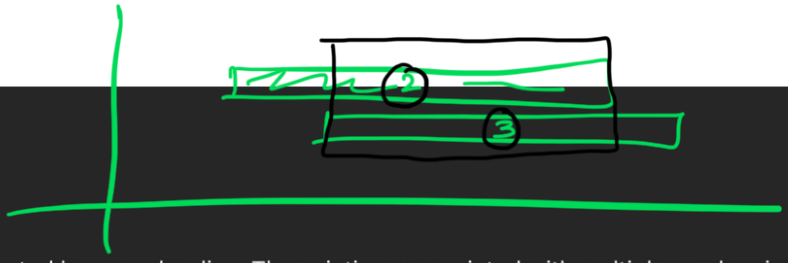
Motivation :

Talent without consistency is like



1943. Describe the Painting

Medium | Topics | Companies | Hint



There is a long and thin painting that can be represented by a number line. The painting was painted with multiple overlapping segments where each segment was painted with a **unique** color. You are given a 2D integer array `segments`, where `segments[i] = [start_i, end_i, color_i]` represents the **half-closed segment** `[start_i, end_i)` with `color_i` as the color.

The colors in the overlapping segments of the painting were **mixed** when it was painted. When two or more colors mix, they form a new color that can be represented as a **set** of mixed colors.

- For example, if colors `2`, `4`, and `6` are mixed, then the resulting mixed color is `{2,4,6}`.

For the sake of simplicity, you should only output the **sum** of the elements in the set rather than the full set.

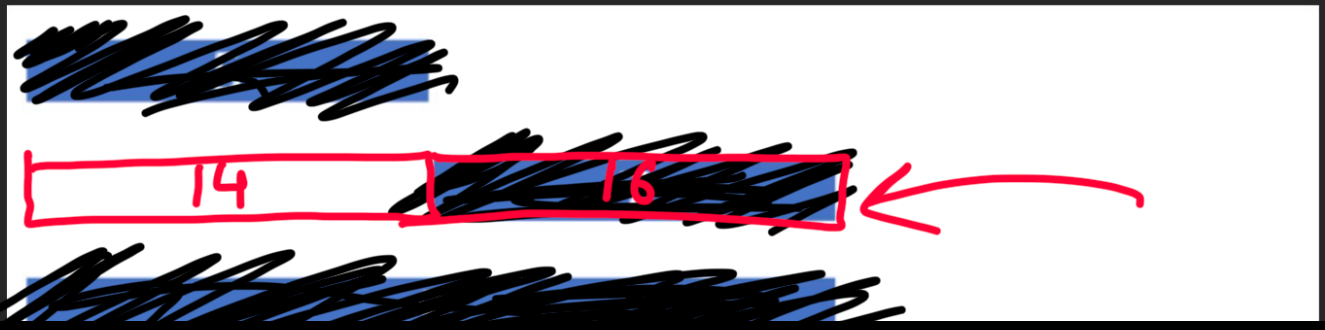
You want to **describe** the painting with the **minimum** number of non-overlapping **half-closed segments** of these mixed colors. These segments can be represented by the 2D array `painting` where `painting[j] = [left_j, right_j, mix_j]` describes a **half-closed segment** `[left_j, right_j)` with the mixed color **sum** of `mix_j`.

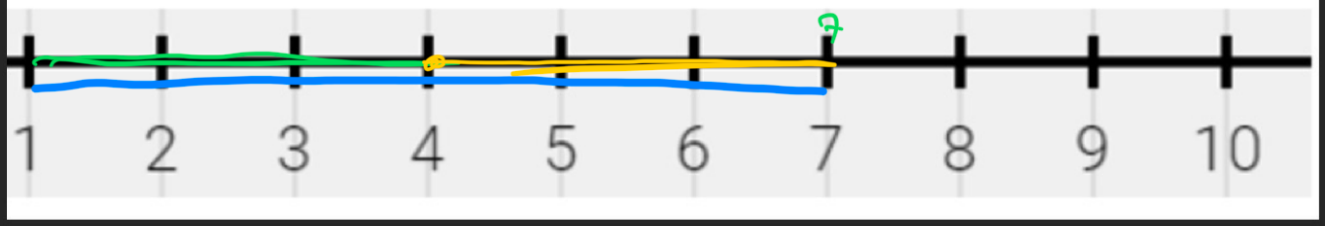
- For example, the painting created with `segments = [[1,4,5],[1,7,7]]` can be described by `painting = [[1,4,12],[4,7,7]]` because:
  - `[1,4)` is colored `{5,7}` (with a sum of `12`) from both the first and second segments.
  - `[4,7)` is colored `{7}` from only the second segment.

Return *the 2D array* `painting` *describing the finished painting (excluding any parts that are* ***not*** *painted). You may return the segments in* ***any order***.

A **half-closed segment** `[a, b)` is the section of the number line between points `a` and `b` **including** point `a` and **not including** point `b`.

Example 1:





Input: segments = [[1,4,5],[4,7,7],[1,7,9]]
Output: [[1,4,14],[4,7,16]]
Explanation: The painting can be described as follows:
- [1,4) is colored {5,9} (with a sum of 14) from the first and third segments.
- [4,7) is colored {7,9} (with a sum of 16) from the second and third segments.

Example 2:

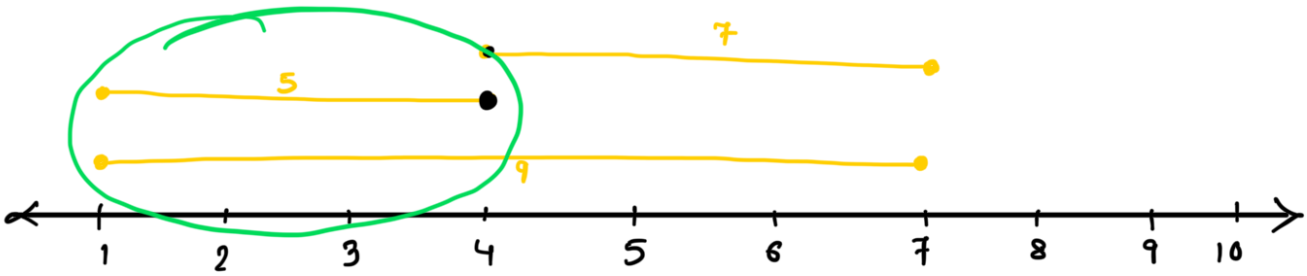
Input: segments = [[1,7,9],[6,8,15],[8,10,7]]
Output: [[1,6,9],[6,7,24],[7,8,15],[8,10,7]]
Explanation: The painting can be described as follows:
- [1,6) is colored 9 from the first segment.
- [6,7) is colored {9,15} (with a sum of 24) from the first and second segments.
- [7,8) is colored 15 from the second segment.
- [8,10) is colored 7 from the third segment.

Constraints:

- $1 <= segments.length <= 2 * 10^4$
- $segments[i].length == 3$ {s, e, c}
- $1 <= start_i < end_i <= 10^5$
- $1 <= color_i <= 10^9$ long long.
- Each $color_i$ is distinct.

Thought Process

nums = { (1, 4, 5), (4, 7, 7), (1, 7, 9) }



(1, 4, 14)

events = { (1, +5), (4, −5), (4, +7), (7, -7), (1, +9), (7, -9) }

sort = { (1, +5), (1, +9), (4, −5), (4, +7), (7, -9), (7, -7) }

map

1	→	14
4	→	2
7	→	-16

→

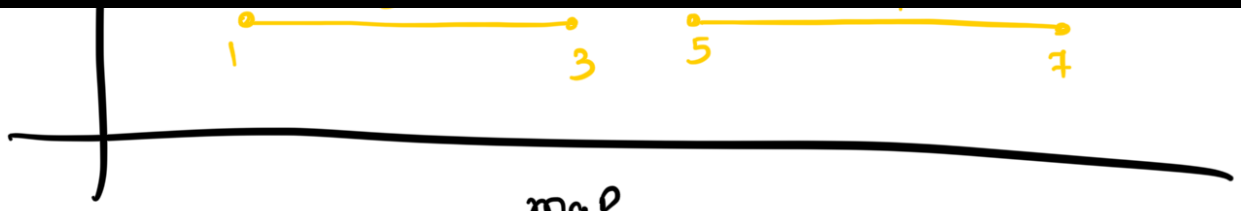
Sum = 0

Start = 7

result = ((1, 4, 14), (4, 7, 16))

(1, 3, 5) (5, 7, 4)

5 4



map

begin → 1 → +5 it;

→ 3 → -5

5 → +4

7 → -4

Start = 7

Sum = 0

if (sum > 0)

{(1, 3, 5), (3, 5, 7), (5, 7, 4)}

Story to Code :-

```
map < int , long long > events;

for ( auto & segment : segments) {
        events [segment[0]] += segment[2];
        events [segment[1]] -= segment[2];
}
```

```
vector<vector<int>> result;
auto it = event.begin();

long long Sum = it->second();
long long start = it->first();

it++;

while (it != events.end()) {
    if (sum > 0) {
        result.push_back((start, it->first, sum));
    }
    start = it->first;
    sum += it->second;

    it++;
}

return result;
```