2021. Brightest Position on Street

Prefix Sum

Ordered Set

Array

Problem Description

Medium

and street lamps are given as a 2D integer array lights. Each entry in this array, lights[i], contains two integers: position_i and range_i. The first integer position_i indicates the location of the street lamp on the number line, while the second integer range_i tells us how far to the left and right the lamp illuminates from its position. Consequently, the area that each street lamp lights up is from position_i - range_i to position_i + range_i inclusive. The problem defines the brightness of a position on the street as the number of street lamps that illuminate that particular position. Our task is to determine the brightest position on the street, which is the position that is lit by the highest number of

In this problem, we are provided with a representation of a street and its street lamps. The street is visualized as a number line,

street lamps. If there happens to be more than one position with the same maximum brightness, the problem asks us to return the smallest of these brightest positions. ntuition

of brightness to a specific range. We can use sweep line algorithm concepts to efficiently determine the brightness of each position. Here's how we can approach the problem:

1. Consider each lamp as triggering an 'event' at the start and end of its range. 2. When a lamp starts at position_i - range_i, it adds to the brightness (+1), and when its influence ends at position_i + range_i + 1, it subtracts from the brightness (-1). This is because the range is inclusive, so the end of the influence is technically one position beyond the

The intuition behind solving this problem involves treating the street as a range of positions and each lamp as contributing a unit

position (ans) where this maximum brightness occurs.

represents positions on the street).

range. 3. We can sweep over all the positions influenced by any lamp, and at each point, we can add or subtract the brightness accordingly. 4. We use a dictionary to keep track of all these 'events' of starting and ending of a lamp's influence, then sort this dictionary by the key (which

5. We sweep the sorted positions and maintain a running sum (s) of the brightness, updating the maximum brightness (mx) seen so far and the

- 6. At the end of the sweep, ans will hold the smallest position with the maximum brightness because we sorted the positions and the calculation respects the order.
- of the brightness changes and keeping track of the running sum and maximum brightness dynamically as we sweep through the positions.

The implementation translates this intuition into a working solution, utilizing Python's defaultdict to facilitate the management

The implementation of the solution uses a variety of data structures and an intelligent algorithm to achieve an efficient approach. **Data Structures Used:**

o defaultdict(int): A Python defaultdict with int type is used to track the starting and ending influence points of each lamp's light on the

street. If an index is not already in the dictionary, it defaults to 0 which is convenient because it represents the initial state of brightness for

through the street, we encounter events that start or end a lamp's range. • Event Processing: Each lamp generates two events: one where its light begins (position_i - range_i) and one where it ends (position_i

street.

position k.

step.

Example Walkthrough

Initialization of Events:

events = defaultdict(int)

Creating and Populating Event Points:

events[5+1] -= 1 # Same as above

Sorting them by key gives us the order [0, 3, 3, 6].

Sweeping Through Sorted Event Points:

influence (s += events[3], now s = 1).

Finding the Brightest Position:

events[2+1] -= 1 # We add 1 because the range end is inclusive

Solution Approach

Algorithms and Patterns: Sweep Line Algorithm: This algorithm is used to simulate the process of 'sweeping' through the street from left to right. As we 'sweep'

+ range_i + 1). In the defaultdict, we increase the brightness by 1 at the start and decrease it by 1 at the end of the range. • Sorting the Events: Sorting the keys of the defaultdict ensures that we process these events in the order of their occurrence along the

any position on the street.

The code reflects these concepts as follows:

• We iterate through each lamp in lights and calculate the starting (1) and ending (r) points of its range, updating the defaultdict accordingly. • We then sort the keys of the defaultdict (the positions where brightness changes occur) and perform the 'sweep' by iteratively adding or subtracting from the running sum s.

• Whenever the running sum s exceeds the previously tracked maximum brightness mx, we update mx with s and set ans to the current

• Running Sum and Maximization: In the sweep process, a running sum s is maintained which updates at each event, increasing or

current sum is greater, we update the mx and record the position ans at which this new maximum occurs.

decreasing according to the lamp's influence. We then compare this running sum to the maximum brightness mx encountered so far. If the

tracks changes in brightness at specific points, which allows us to find the brightest position without redundant calculations.

As a result, the code does not require us to calculate the brightness for every single position on the street. Instead, it efficiently

Let's go through the solution approach with a small example. Suppose we are given the following lights array representing the

Each of the subarrays represents a street lamp with [position_i, range_i]. Now, we will walk through the algorithm step by

street lamps: lights = [[1, 1], [4, 1]]

• For the first lamp [1, 1], it influences the street in the range [0, 2] (from position_i - range_i to position_i + range_i). • For the second lamp [4, 1], it influences the street in the range [3, 5].

• We treat the start and end of each range as events. We use a defaultdict to store the brightness increase and decrease at specific points:

Sorting Events: We now have the events {0: 1, 3: 1, 3: −1, 6: −1}.

our final answer.

from collections import defaultdict

Iterate over each light

for position, radius in lights:

def brightestPosition(self, lights: List[List[int]]) -> int:

brightness_changes = defaultdict(int)

left border = position - radius

right_border = position + radius

brightness changes[left border] += 1

for position in sorted(brightness changes):

if max brightness < current brightness:</pre>

brightest_position = position

max brightness = current brightness

Update the current brightness

public int brightestPosition(int[][] lights) {

return brightest_position

for (int[] light : lights) {

int brightestPosition = 0:

int currentBrightness = 0:

return brightestPosition;

};

TypeScript

currentBrightness += delta;

int maxBrightness = 0;

// Variable to keep track of the current sum of brightness as we iterate

// If we find a brighter position, update maxBrightness and brightestPosition

// Iterate through brightnessDeltas to find the brightest position

// Variable to store the maximum brightness encountered

// Update the current brightness based on the delta

for (auto& [position, delta] : brightnessDeltas) {

if (maxBrightness < currentBrightness) {</pre>

brightestPosition = position;

maxBrightness = currentBrightness;

// Return the position with the maximum brightness

brightness changes[right_border + 1] -= 1

Initialize a dictionary to store the changes in brightness

Increment the brightness at the start of the range

Decrement the brightness just after the end of the range

Initialize variables to keep track of the current brightness,

current brightness += brightness changes[position]

TreeMap<Integer, Integer> deltaBrightness = new TreeMap<>();

Determine the range of positions affected by the light's brightness

If the current brightness is greater than the maximum recorded brightness

// Use a TreeMap to easily manage the range of light contributions on the positions

// Iterate over each light array to calculate the influence ranges and store them

int leftBoundary = light[0] - light[1]; // Calculate left boundary of the light

int rightBoundary = light[0] + light[1]; // Calculate right boundary of the light

Update the maximum brightness and brightest position

Python

class Solution:

events[0] += 1

events[3] += 1

- \circ At position 0, we encounter the start of the first lamp's influence, so s += events[0] (now s = 1). At position 3, we have two events. The start of the second lamp's influence (s += events[3], now s = 2) and the end of the first lamp's
- decreases back to 1 by the end of processing all events at 3. ans during the occurrence of mx is set to the position at which this happened, which in this case, initially, is 3.

 \circ At position 6, we encounter the end of the second lamp's influence, so $s \rightarrow events[6]$ (now s = 0).

After processing each event, we update mx and ans whenever we find a new maximum brightness.

• We begin with a running sum s = 0, maximum brightness mx = 0, and the answer ans = 0.

which makes the algorithm efficient and effective for solving this kind of problem. Solution Implementation

 \circ During the process, we found the maximum brightness to be mx = 2 at positions 3, but since there are two events at 3, this number

Hence, by the end of the sweep, we determined that the brightest position on the street is 3 with a brightness of 2, and that is

By following this approach, we only calculate the brightness at specific event points instead of for every position on the street,

the maximum brightness observed, and the position of the maximum brightness current brightness = max_brightness = 0 brightest_position = 0 # Iterate over the positions in the sorted order of the keys

```
# Return the position with the maximum brightness
```

class Solution {

Java

```
// Increase brightness at the start of the range
            deltaBrightness.merge(leftBoundary, 1, Integer::sum);
            // Decrease brightness right after the end of the range
            deltaBrightness.merge(rightBoundary + 1, -1, Integer::sum);
        int brightestPosition = 0: // To hold the result position with the brightest light
        int currentBrightness = 0; // Current accumulated brightness
        int maxBrightness = 0; // Max brightness observed at any point
        // Iterate over the entries in the TreeMap
        for (var entry : deltaBrightness.entrySet()) {
            int changeInBrightness = entry.getValue();
            currentBrightness += changeInBrightness; // Apply the change on the current brightness
            // Check if the current brightness is the maximum observed so far
            if (maxBrightness < currentBrightness) {</pre>
                maxBrightness = currentBrightness; // Update the maximum brightness
                brightestPosition = entry.getKey(); // Update the position of the brightest light
        return brightestPosition; // Return the position with the maximum brightness
C++
class Solution {
public:
    int brightestPosition(vector<vector<int>>& lights) {
        // Create a map to store the changes in brightness at different positions
        map<int, int> brightnessDeltas;
        // Loop through each light and update the map with the range it illuminates
        for (auto& light: lights) {
            // Calculate the left and right bounds of the light's effect
            int left = light[0] - light[1];
            int right = light[0] + light[1];
            // Increase brightness at the starting position of the light's effect
            ++brightnessDeltas[left]:
            // Decrease brightness just after the end of the light's effect
             --brightnessDeltas[right + 1];
        // Variable to store the brightest position found so far
```

```
// Define the type structure for the light positions array
type LightPosition = [number, number];
/**
* Returns the brightest position on a street given the array of lights.
 * @param lights - An array of tuples representing lights, where each tuple consists of the position and range.
 * @return The position of the brightest point.
const brightestPosition = (lights: LightPosition[]): number => {
   // Create a map where the key is the position on the street and the value is the change in brightness at that point.
   const deltaBrightness = new Map<number, number>();
   // Populate the map with brightness changes, accounting for the lights turning on at their start position
   // and off immediately after their end position.
    for (const [position, range] of lights) {
        const start = position - range;
        const end = position + range;
       deltaBrightness.set(start, (deltaBrightness.get(start) ?? 0) + 1);
       deltaBrightness.set(end + 1, (deltaBrightness.get(end + 1) ?? 0) - 1);
   // Extract and sort the keys from the map to iterate over positions in ascending order.
   const sortedPositions = Array.from(deltaBrightness.keys()).sort((a, b) => a - b);
    let currentBrightness = 0; // Accumulated brightness at the current position.
    let maxBrightness = 0;  // Maximum brightness encountered so far.
    let brightestPos = 0;  // Position with the maximum brightness.
   // Iterate over all positions to find the maximum accumulated brightness.
    for (const position of sortedPositions) {
        currentBrightness += deltaBrightness.get(position) || 0;
       // Update maximum brightness and position if the current brightness is greater.
       if (maxBrightness < currentBrightness) {</pre>
            maxBrightness = currentBrightness;
            brightestPos = position;
   // Return the position with the maximum brightness.
   return brightestPos;
// Example usage (Optional):
// const lights: LightPosition[] = [[1, 2], [3, 6], [5, 5]];
// console.log(brightestPosition(lights)); // Should output the position of the brightest point
from collections import defaultdict
class Solution:
   def brightestPosition(self, lights: List[List[int]]) -> int:
       # Initialize a dictionary to store the changes in brightness
```

Return the position with the maximum brightness return brightest_position

brightness_changes = defaultdict(int)

left border = position - radius

current brightness = max_brightness = 0

Update the current brightness

for position in sorted(brightness changes):

if max brightness < current brightness:</pre>

brightest_position = position

max brightness = current brightness

right_border = position + radius

brightness changes[left border] += 1

brightness_changes[right_border + 1] -= 1

Determine the range of positions affected by the light's brightness

the maximum brightness observed, and the position of the maximum brightness

If the current brightness is greater than the maximum recorded brightness

Update the maximum brightness and brightest position

Increment the brightness at the start of the range

Decrement the brightness just after the end of the range

Initialize variables to keep track of the current brightness,

Iterate over the positions in the sorted order of the keys

current brightness += brightness changes[position]

Iterate over each light

brightest_position = 0

Time and Space Complexity

for position, radius in lights:

The time complexity of the code is O(N log N) where N is the number of light ranges in the lights list. This complexity arises because we sort the keys of our dictionary d, which contains at most 2N keys (each light contributes two keys: the start and end of its illumination range). Sorting these keys dominates the runtime complexity.

The space complexity of the code is O(N) since we use a dictionary to store the changes to brightness at each key point. In the worst case, if every light has a unique range, the dictionary could have as many as 2N keys, where N is the number of light ranges in the lights list.