683. K Empty Slots

Sept. 27, 2017 | 34.9K views

Average Rating: 2.23 (56 votes)

You have `N` bulbs in a row numbered from `1` to `N`. Initially, all the bulbs are turned off. We turn on exactly one bulb everyday until all bulbs are on after `N` days.

You are given an array `bulbs` of length `N` where `bulbs[i] = x` means that on the `(i+1)th` day, we will turn on the bulb at position `x` where `i` is `0-indexed` and `x` is `1-indexed.`

Given an integer `K`, find out the **minimum day number** such that there exists two **turned on** bulbs that have **exactly** `K` bulbs between them that are **all turned off**.

If there isn't such day, return `-1`.

Example 1:

```
Input:
bulbs: [1,3,2]
K: 1
Output: 2
Explanation:
On the first day: bulbs[0] = 1, first bulb is turned on: [1,0,0]
On the second day: bulbs[1] = 3, third bulb is turned on: [1,0,1]
On the third day: bulbs[2] = 2, second bulb is turned on: [1,1,1]
We return 2 because on the second day, there were two on bulbs with one off bulb betwe
```

Example 2:

```
Input:
bulbs: [1,2,3]
K: 1
Output: -1
```

Note:

1. `1 <= N <= 20000`
2. `1 <= bulbs[i] <= N`
3. `bulbs` is a permutation of numbers from `1` to `N`.
4. `0 <= K <= 20000`

Approach #1: Insert Into Sorted Structure [Accepted]

Intuition

Let's add flowers in the order they bloom. When each flower blooms, we check it's neighbors to see if they can satisfy the condition with the current flower.

Algorithm

We'll maintain `active`, a sorted data structure containing every flower that has currently bloomed. When we add a flower to `active`, we should check it's lower and higher neighbors. If some neighbor satisfies the condition, we know the condition occurred first on this day.

Java Python Copy

```python
class Solution(object):
    def kEmptySlots(self, flowers, k):
        active = []
        for day, flower in enumerate(flowers, 1):
            i = bisect.bisect(active, flower)
            for neighbor in active[i-(i>0):i+1]:
                if abs(neighbor - flower) - 1 == k:
                    return day
            active.insert(i, flower)
        return -1
```

Complexity Analysis

- Time Complexity (Java): $O(N \log N)$, where N is the length of `flowers`. Every insertion and search is $O(\log N)$.
- Time Complexity (Python): $O(N^2)$. As above, except `list.insert` is $O(N)$.
- Space Complexity: $O(N)$, the size of `active`.

Approach #2: Min Queue [Accepted]

Intuition

For each contiguous block ("window") of `k` positions in the flower bed, we know it satisfies the condition in the problem statement if the minimum blooming date of this window is larger than the blooming date of the left and right neighbors.

Because these windows overlap, we can calculate these minimum queries more efficiently using a sliding window structure.

Algorithm

Let `days[x] = i` be the time that the flower at position `x` blooms. For each window of `k` days, let's query the minimum of this window in (amortized) constant time using a `MinQueue`, a data structure built just for this task. If this minimum is larger than it's two neighbors, then we know this is a place where "`k` empty slots" occurs, and we record this candidate answer.

To operate a `MinQueue`, the key invariant is that `mins` will be an increasing list of candidate answers to the query `MinQueue.min`.

For example, if our queue is `[1, 3, 6, 2, 4, 8]`, then `mins` will be `[1, 2, 4, 8]`. As we `MinQueue.popleft`, `mins` will become `[2, 4, 8]`, then after 3 more `popleft`'s will become `[4, 8]`, then after 1 more `popleft` will become `[8]`.

As we `MinQueue.append`, we should maintain this invariant. We do it by popping any elements larger than the one we are inserting. For example, if we appended `5` to `[1, 3, 6, 2, 4, 8]`, then `mins` which was `[1, 2, 4, 8]` becomes `[1, 2, 4, 5]`.

Note that we used a simpler variant of `MinQueue` that requires every inserted element to be unique to ensure correctness. Also, the operations are amortized constant time because every element will be inserted and removed exactly once from each queue.

Java Python Copy

```python
from collections import deque
class MinQueue(deque):
    def __init__(self):
        deque.__init__(self)
        self.mins = deque()

    def append(self, x):
        deque.append(self, x)
        while self.mins and x < self.mins[-1]:
            self.mins.pop()
        self.mins.append(x)

    def popleft(self):
        x = deque.popleft(self)
        if self.mins[0] == x:
            self.mins.popleft()
        return x

    def min(self):
        return self.mins[0]

class Solution(object):
    def kEmptySlots(self, flowers, k):
        days = [0] * len(flowers)
        for day, position in enumerate(flowers, 1):
            days[position - 1] = day

        window = MinQueue()
```

Complexity Analysis

- Time Complexity: $O(N)$, where N is the length of `flowers`. In enumerating through the $O(N)$ outer loop, we do constant work as `MinQueue.popleft` and `MinQueue.min` operations are (amortized) constant time.
- Space Complexity: $O(N)$, the size of our `window`.

Approach #3: Sliding Window [Accepted]

Intuition

As in *Approach #2*, we have `days[x] = i` for the time that the flower at position `x` blooms. We wanted to find *candidate* intervals `[left, right]` where `days[left], days[right]` are the two smallest values in `[days[left], days[left+1], ..., days[right]]`, and `right - left = k + 1`.

Notice that these candidate intervals cannot intersect: for example, if the candidate intervals are `[left1, right1]` and `[left2, right2]` with `left1 < left2 < right1 < right2`, then for the first interval to be a candidate, `days[left2] > days[right1]`; and for the second interval to be a candidate, `days[right1] > days[left2]`, a contradiction.

That means whenever whether some interval can be a candidate and it fails first at `i`, indices `j < i` can't be the start of a candidate interval. This motivates a sliding window approach.

Algorithm

As in *Approach #2*, we construct `days`.

Then, for each interval `[left, right]` (starting with the first available one), we'll check whether it is a candidate: whether `days[i] > days[left]` and `days[i] > days[right]` for `left < i < right`.

If we fail, then we've found some new minimum `days[i]` and we should check the new interval `[i, i+k+1]`. If we succeed, then it's a candidate answer, and we'll check the new interval `[right, right+k+1]`.

Java Python Copy

```python
class Solution(object):
    def kEmptySlots(self, flowers, k):
        days = [0] * len(flowers)
        for day, position in enumerate(flowers, 1):
            days[position - 1] = day

        ans = float('inf')
        left, right = 0, k+1
        while right < len(days):
            for i in xrange(left + 1, right):
                if days[i] < days[left] or days[i] < days[right]:
                    left, right = i, i+k+1
                    break
            else:
                ans = min(ans, max(days[left], days[right]))
                left, right = right, right+k+1

        return ans if ans < float('inf') else -1
```

Complexity Analysis

- Time and Space Complexity: $O(N)$. The analysis is the same as in Approach #2.

Analysis written by: @awice. Approach #1 inspired by @StefanPochmann. Approach #3 inspired by @Vincent Cai.

Rate this article:

Previous Next

Comments: 35 Sort By

Type comment here... (Markdown is supported)

Preview Post

MichaelrMentele ★ 88 April 4, 2019 7:41 AM

Anyone else find the problem statement for this question incredibly confusing?

69 Share Reply

SHOW 1 REPLY

dongseong ★ 71 October 14, 2018 1:12 AM

omg, it takes 3days to understand 2nd solution :'(

21 Share Reply

SHOW 1 REPLY

hai_chen ★ 13 April 14, 2018 1:04 AM

I have got one test case failure says the correct answer for {6,5,8,9,7,1,10,2,3,4} k=2 is 8, shouldn't it be 7 ?

after 7 days X X _ _ X X _ X X X , shed some light on this please.

13 Share Reply

SHOW 2 REPLIES

addoil ★ 55 November 14, 2018 10:30 PM

In Approach 2, should the time complexity be O(Nlog(k))? there are N operations in outer loop, in each loop, the add and poll operation is log(k), where k is the size of heap?

10 Share Reply

SHOW 1 REPLY

OneQi ★ 10 May 5, 2018 10:45 PM

Could you please avoid using goto in java?

10 Share Reply

SHOW 2 REPLIES

novice87 ★ 226 July 15, 2019 3:19 AM

The problem description is regarding bulbs, and the solutions are written for flower blooming. Can we modify the problem description to avoid the confusion?

9 Share Reply

KaiPeng21 ★ 749 February 3, 2019 5:19 AM

I be honest, don't like the way leetcode wrote certain python solutions. Shorter but less readable...

Here's solution 1 using Python

```
        active = []
```

Read More

8 Share Reply

SHOW 1 REPLY

dhruvmonga ★ 2 April 17, 2019 6:32 AM

Solution 2 and 3 are wrong.
The following block:
for day, position in enumerate(flowers, 1):
days[position - 1] = day
Should be:

Read More

2 Share Reply

XiangkunYe ★ 60 November 5, 2018 4:56 PM

Python solution of Approach #2 cannot work, just copy it and submit and you will see.
In the last line:

```
return ans if ans <= len(days) else -1
```

Read More

2 Share Reply

littlefinzer ★ 5 November 27, 2017 10:42 PM

Can we add 1 more approach to an existing solution article? There's another way for sliding window but involves less steps than using a min queue.

2 Share Reply

SHOW 1 REPLY

‹ 1 2 3 4 ›