K Empty Slots

There is a garden with `N` slots. In each slot, there is a flower. The `N` flowers will bloom one by one in `N` days. In each day, there will be `exactly` one flower blooming and it will be in the status of blooming since then.

Given an array `flowers` consists of number from `1` to `N`. Each number in the array represents the place where the flower will open in that day.

For example, `flowers[i] = x` means that the unique flower that blooms at day `i` will be at position `x`, where `i` and `x` will be in the range from `1` to `N`.

Also given an integer `k`, you need to output in which day there exists two flowers in the status of blooming, and also the number of flowers between them is `k` and these flowers are not blooming.

If there isn't such day, output -1.

Example 1:

```
Input:
flowers: [1,3,2]
k: 1
Output: 2
Explanation: In the second day, the first and the third flower have become blooming.
```

Example 2:

```
Input:
flowers: [1,2,3]
k: 1
Output: -1
```

Note:

1. The given array will be in the range [1, 20000].

Solution 1

It seems that this question has some mistakes. I think there are two places that might lead to misunderstandings: (please feel free to tell me if I'm incorrect)

1. `flowers[i] = x` should mean that the unique flower that blooms at day `i+1` (not `i`) will be at position `x`.
2. If you can get multiple possible results, then you need to return the minimum one.

The idea is to use an array `days[]` to record each position's flower's blooming day. That means `days[i]` is the blooming day of the flower in position `i+1`. We just need to find a subarray `days[left, left+1,..., left+k-1, right]` which satisfies: for any `i = left+1,..., left+k-1`, we can have `days[left] < days[i] && days[right] < days[i]`. Then, the result is `max(days[left], days[right])`.

Java version:

```
public int kEmptySlots(int[] flowers, int k) {
        int[] days =  new int[flowers.length];
        for(int i=0; i<flowers.length; i++)days[flowers[i] - 1] = i + 1;
        int left = 0, right = k + 1, res = Integer.MAX_VALUE;
        for(int i = 0; right < days.length; i++){
            if(days[i] < days[left] || days[i] <= days[right]){
                if(i == right)res = Math.min(res, Math.max(days[left], days[right])
);   //we get a valid subarray
                left = i;
                right = k + 1 + i;
            }
        }
        return (res == Integer.MAX_VALUE)?-1:res;
    }
```

C++ version:

```
int kEmptySlots(vector<int>& flowers, int k) {
        vector<int> days(flowers.size());
        for(int i=0; i<flowers.size();i++)days[flowers[i] - 1] = i + 1;
        int left = 0, right = k + 1, res = INT_MAX;
        for(int i = 0; right < days.size(); i++){
            if(days[i] < days[left] || days[i] <= days[right]){
                if(i == right)res = min(res, max(days[left], days[right]));    //we
get a valid subarray
                left = i, right = k + 1 + i;
            }
        }
        return (res == INT_MAX)?-1:res;
    }
```

written by Vincent Cai original link here

Solution 2

```
public int kEmptySlots(int[] flowers, int k) {
 TreeSet<Integer> treeSet = new TreeSet<>();
 for (int i = 0; i < flowers.length; i++) {
  int current = flowers[i];
  Integer next = treeSet.higher(current);
  if (next != null && next - current == k + 1) {
   return i + 1;
  }
  Integer pre = treeSet.lower(current);
  if (pre != null && current - pre == k + 1) {
   return i + 1;
  }
  treeSet.add(current);
 }
 return -1;
}
```

written by navid original link here

Solution 3

```
class Solution {
public:
    int kEmptySlots(vector<int>& flowers, int k) {
        set<int> bloom;
        for (int i = 0; i < flowers.size(); i++) {
            int p = flowers[i];
            auto it = bloom.insert(p).first;
            if (it != bloom.begin()) {
                if (p-*(--it) == k+1) return i+1;
                it++;
            }
            if (++it != bloom.end() && *it-p == k+1) return i+1;
        }
        return -1;
    }
};
```

written by zestypanda original link here