2612. Minimum Reverse Operations

Description

You are given an integer `n` and an integer `p` in the range `[0, n - 1]`. Representing a **0-indexed** array `arr` of length `n` where all positions are set to `0`'s, except position `p` which is set to `1`.

You are also given an integer array `banned` containing some positions from the array. For the **i** th position in `banned`, `arr[banned[i]] = 0`, and `banned[i] != p`.

You can perform **multiple** operations on `arr`. In an operation, you can choose a **subarray** with size `k` and **reverse** the subarray. However, the `1` in `arr` should never go to any of the positions in `banned`. In other words, after each operation `arr[banned[i]]` **remains** `0`.

Return an array `ans` *where for each* `i` *from* `[0, n - 1]`, `ans[i]` *is the* ***minimum*** *number of reverse operations needed to bring the* `1` *to position* `i` *in arr, or* `-1` *if it is impossible*.

- A **subarray** is a contiguous **non-empty** sequence of elements within an array.
- The values of `ans[i]` are independent for all `i`'s.
- The **reverse** of an array is an array containing the values in **reverse order**.

Example 1:

```
Input: n = 4, p = 0, banned = [1,2], k = 4
Output: [0,-1,-1,1]
Explanation: In this case k = 4 so there is only one possible reverse operation we can perform, which is reversing the whole array. Initially, 1 is placed at position 0 so the amount of operations we need for position 0 is 0. We can never place a 1 on the banned positions, so the answer for positions 1 and 2 is -1. Finally, with one reverse operation we can bring the 1 to index 3, so the answer for position 3 is 1.
```

Example 2:

```
Input: n = 5, p = 0, banned = [2,4], k = 3
Output: [0,-1,-1,-1,-1]
Explanation: In this case the 1 is initially at position 0, so the answer for that position is 0. We can perform reverse operations of size 3. The 1 is currently located at position 0, so we need to reverse the subarray [0, 2] for it to leave that position, but reversing that subarray makes position 2 have a 1, which shouldn't happen. So, we can't move the 1 from position 0, making the result for all the other positions -1.
```

Example 3:

```
Input: n = 4, p = 2, banned = [0,1,3], k = 1
Output: [-1,-1,0,-1]
Explanation: In this case we can only perform reverse operations of size 1.  So the 1 never changes its position.
```

Constraints:

- `1 <= n <= 10 ^ 5`
- `0 <= p <= n - 1`
- `0 <= banned.length <= n - 1`
- `0 <= banned[i] <= n - 1`
- `1 <= k <= n`
- `banned[i] != p`
- all values in `banned` are **unique**