Description | Solution | Discuss (242) | Submissions

1151. Minimum Swaps to Group All 1's Together

Medium 550 5 Add to List Share

Given a binary array `data`, return the minimum number of swaps required to group all `1`'s present in the array together in **any place** in the array.

Example 1:

```
Input: data = [1,0,1,0,1]
Output: 1
Explanation:
There are 3 ways to group all 1's together:
[1,1,1,0,0] using 1 swap.
[0,1,1,1,0] using 2 swaps.
[0,0,1,1,1] using 1 swap.
The minimum is 1.
```

Example 2:

```
Input: data = [0,0,0,1,0]
Output: 0
Explanation:
Since there is only one 1 in the array, no swaps needed.
```

Example 3:

```
Input: data = [1,0,1,0,1,0,0,1,1,0,1]
Output: 3
Explanation:
One possible solution that uses 3 swaps is [0,0,0,0,0,1,1,1,1,1,1].
```

Example 4:

```
Input: data =
[1,0,1,0,1,0,1,1,1,0,1,0,0,1,1,1,0,0,1,1,1,0,1,0,1,1,0,1,0,0,0,1,1,1,1,0,0,1]
Output: 8
```

Constraints:

- `1 <= data.length <= 10^5`
- `data[i]` is `0` or `1`.

Accepted 21,675 | Submissions 36,759

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months 6 months ~ 1 year 1 year ~ 2 years

Amazon | 10

Related Topics

Array Sliding Window

Similar Questions

Minimum Adjacent Swaps for K Consecutive Ones Hard

Hide Hint 1

How many 1's should be grouped together ? Is not a fixed number?

Hide Hint 2

Yeah it's just the number of 1's the whole array has. Let's name this number as ones

Hide Hint 3

Every subarray of size of ones, needs some number of swaps to reach, Can you find the number of swaps needed to group all 1's in this subarray?

Hide Hint 4

It's the number of zeros in that subarray.

Hide Hint 5

Do you need to count the number of zeros all over again for every position ?

Hide Hint 6

Use Sliding Window technique.

Java | Autocomplete

```java
class Solution {
    public int minSwaps(int[] data) {
        int ones = Arrays.stream(data).sum();
        int cnt_one = 0, max_one = 0;
        // maintain a deque with the size = ones
        Deque<Integer> deque = new ArrayDeque<>();

        for (int i = 0; i < data.length; i++) {

            // we would always add the new element into the deque
            deque.addLast(data[i]);
            cnt_one += data[i];

            // when there are more than ones elements in the deque,
            // remove the leftmost one
            if (deque.size() > ones) {
                cnt_one -= deque.removeFirst();;
            }
            max_one = Math.max(max_one, cnt_one);
        }
        return ones - max_one;

    }
}
```