Minimum Swaps to Group All 1's Together II

Description:

A swap is defined as taking two distinct positions in an array and swapping the values in them.

A circular array is defined as an array where we consider the first element and the last element to be adjacent.

Given a binary circular array nums, return *the minimum number of swaps required to group all* 1*'s present in the array together at any location*.

Example 1:

Input: nums = [0,1,0,1,1,0,0]

Output: 1

Explanation: Here are a few of the ways to group all the 1's together:

[0,0,1,1,1,0,0] using 1 swap.

[0,1,1,1,0,0,0] using 1 swap.

[1,1,0,0,0,0,1] using 2 swaps (using the circular property of the array).

There is no way to group all 1's together with 0 swaps.

Thus, the minimum number of swaps required is 1.

Example 2:

Input: nums = [0,1,1,1,0,0,1,1,0]

Output: 2

Explanation: Here are a few of the ways to group all the 1's together:

[1,1,1,0,0,0,0,1,1] using 2 swaps (using the circular property of the array).

[1,1,1,1,1,0,0,0,0] using 2 swaps.

There is no way to group all 1's together with 0 or 1 swaps.

Thus, the minimum number of swaps required is 2.

Example 3:

Input: nums = [1,1,0,0,1]

Output: 0

Explanation: All the 1's are already grouped together due to the circular property of the array.

Thus, the minimum number of swaps required is 0.

Algorithm

1.) **Calculate the Window Size**:

- Iterate over the nums array to count the total number of 1's. This count will be the size of the sliding window.

2.) **Count Zeros in the Initial Window**:

- Iterate over the first windowSize elements of the nums array to count the number of zeros in this initial window.

3.) **Initialize Variables**:

- curZeros to keep track of the current number of zeros in the current window.
- minZeros to store the minimum number of zeros found in any window, initialized to curZeros.
- start to indicate the start of the current window.
- end to indicate the end of the current window (initialized to windowSize - 1).

4.) **Slide the Window**:

- Use a while loop to slide the window across the array.
- If the element at the start position of the window is zero, decrement curZeros.
- Increment start to move the window to the right.
- Increment end and use modulo n to wrap around the array (circular behavior).
- If the new element at the end position is zero, increment curZeros.
- Update minZeros to be the minimum of minZeros and curZeros.

5.) **Return the Result**:

- The result is minZeros, which represents the minimum number of zeros in any window of size equal to the number of 1's, and thus the minimum number of swaps needed.

Pseudocode

```
function minSwaps(nums: array of int) -> int:
    windowSize = sum(nums)  # Count of 1's in the array
    curZeros = 0

    # Count zeros in the initial window
    for i from 0 to windowSize - 1:
        if nums[i] == 0:
            curZeros += 1

    minZeros = curZeros
    start = 0
    end = windowSize - 1
    n = length of nums

    while start < n:
        if nums[start] == 0:
            curZeros -= 1
        start += 1

        end += 1
        if nums[end % n] == 0:
            curZeros += 1

        minZeros = min(minZeros, curZeros)

    return minZeros
```

Code

```
class Solution {
    public int minSwaps(int[] nums) {
        // window size - count of 1
        int windowSize = 0;
        for(int num:nums){
            windowSize += num;
        }
        // Find zeros in first window
        int curZeros = 0;
        for(int i = 0; i<windowSize; i++){
            if(nums[i] == 0){
                curZeros++;
            }
        }
        // Solve for remaining window
        int minZeros = curZeros;
        int start = 0;
        int end = windowSize - 1;
        int n = nums.length;
        while(start < n){
            // if removed element was 0, decrement 0 counter
            if(nums[start]==0){
                curZeros--;
            }
            start++;
            // if included element is 0, increment 0 counter
            end++;
            if(nums[end % n] == 0){
```

```
            curZeros++;
          }
          minZeros = Math.min(minZeros, curZeros);
        }
        return minZeros;
    }
}
```

Conclusion

The minSwaps function efficiently calculates the minimum number of swaps needed to group all 1's together in a circular array. It starts by determining the window size, which is equal to the number of 1's in the array, and then counts the zeros in the initial window of this size. Using a sliding window technique, the function traverses the array circularly, adjusting the zero count dynamically for each window. By keeping track of the minimum number of zeros encountered, it identifies the optimal window that requires the fewest swaps to group all 1's together. The result, minZeros, represents this minimum number of swaps, ensuring that the 1's are grouped with the least amount of movement in the array.