MW

Longest Subarray with Ones after Replacement (hard)

Problem Statement

Given an array containing 0s and 1s, if you are allowed to **replace no more than 'k' 0s with 1s**, find the length of the **longest contiguous subarray having all 1s**.

Example 1:

```
Input: Array=[0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1], k=2
Output: 6
Explanation: Replace the '0' at index 5 and 8 to have the longest contiguous subarray of 1s hav
ing length 6.
```

Example 2:

```
Input: Array=[0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1], k=3
Output: 9
Explanation: Replace the '0' at index 6, 9, and 10 to have the longest contiguous subarray of 1
s having length 9.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const length_of_longest_substring = function(arr, k) {
  // TODO: Write your code here
  return -1;
};

```

TEST SAVE RESET

Solution

This problem follows the **Sliding Window** pattern and is quite similar to Longest Substring with same Letters after Replacement. The only difference is that, in the problem, we only have two characters (1s and 0s) in the input arrays.

Following a similar approach, we'll iterate through the array to add one number at a time in the window. We'll also keep track of the maximum number of repeating 1s in the current window (let's call it `maxOnesCount`). So at any time, we know that we can have a window which has 1s repeating `maxOnesCount` time, so we should try to replace the remaining 0s. If we have more than 'k' remaining 0s, we should shrink the window as we are not allowed to replace more than 'k' 0s.

Code

Here is how our algorithm will look like:

Java | Python3 | C++ | JS

```
function length_of_longest_substring(arr, k) {
  let windowStart = 0,
    maxLength = 0,
    maxOnesCount = 0;

  // Try to extend the range [windowStart, windowEnd]
  for (windowEnd = 0; windowEnd < arr.length; windowEnd++) {
    if (arr[windowEnd] === 1) {
      maxOnesCount += 1;
    }

    // Current window size is from windowStart to windowEnd, overall we have a maximum of 1s
    // repeating 'maxOnesCount' times, this means we can have a window with 'maxOnesCount' 1s
    // and the remaining are 0s which should replace with 1s.
    // now, if the remaining 1s are more than 'k', it is the time to shrink the window as we
    // are not allowed to replace more than 'k' 0s
    if ((windowEnd - windowStart + 1 - maxOnesCount) > k) {
      if (arr[windowStart] === 1) {
        maxOnesCount -= 1;
      }
```

Explore
Tracks
My Courses
Edpresso
Refer a Friend

Mark Course as Completed
Create

```
      windowStart += 1;
    }

    maxLength = Math.max(maxLength, windowEnd - windowStart + 1);
  }
  return maxLength;
}

```

RUN SAVE RESET

Time Complexity

The time complexity of the above algorithm will be $O(N)$ where 'N' is the count of numbers in the input array.

Space Complexity

The algorithm runs in constant space $O(1)$.

☑ MARK AS COMPLETED

← Back
Longest Substring with Same Letters ...

Next →
Problem Challenge 1

Report an Issue