


525. Contiguous Array

March 24, 2017 | 231K views

PreviousNext


Average Rating: 4.53 (100 votes)

Given a binary array, find the maximum length of a contiguous subarray with equal number of 0 and 1.

Example 1:

Input: [0,1]
Output: 2
Explanation: [0, 1] is the longest contiguous subarray with equal number of 0 and 1.

Example 2:

Input: [0,1,0]
Output: 2
Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of

Note: The length of the given binary array will not exceed 50,000.

Solution

Approach #1 Brute Force [Time Limit Exceeded]

Algorithm

The brute force approach is really simple. We consider every possible subarray within the given array and count the number of zeros and ones in each subarray. Then, we find out the maximum size subarray with equal no. of zeros and ones out of them.

JavaCopy

```
1
2 public class Solution {
3
4     public int findMaxLength(int[] nums) {
5         int maxlen = 0;
6         for (int start = 0; start < nums.length; start++) {
7             int zeroes = 0, ones = 0;
8             for (int end = start; end < nums.length; end++) {
9                 if (nums[end] == 0) {
10                     zeroes++;
11                 } else {
12                     ones++;
13                 }
14                 if (zeroes == ones) {
15                     maxlen = Math.max(maxlen, end - start + 1);
16                 }
17             }
18         }
19         return maxlen;
20     }
21 }
22
```

Complexity Analysis

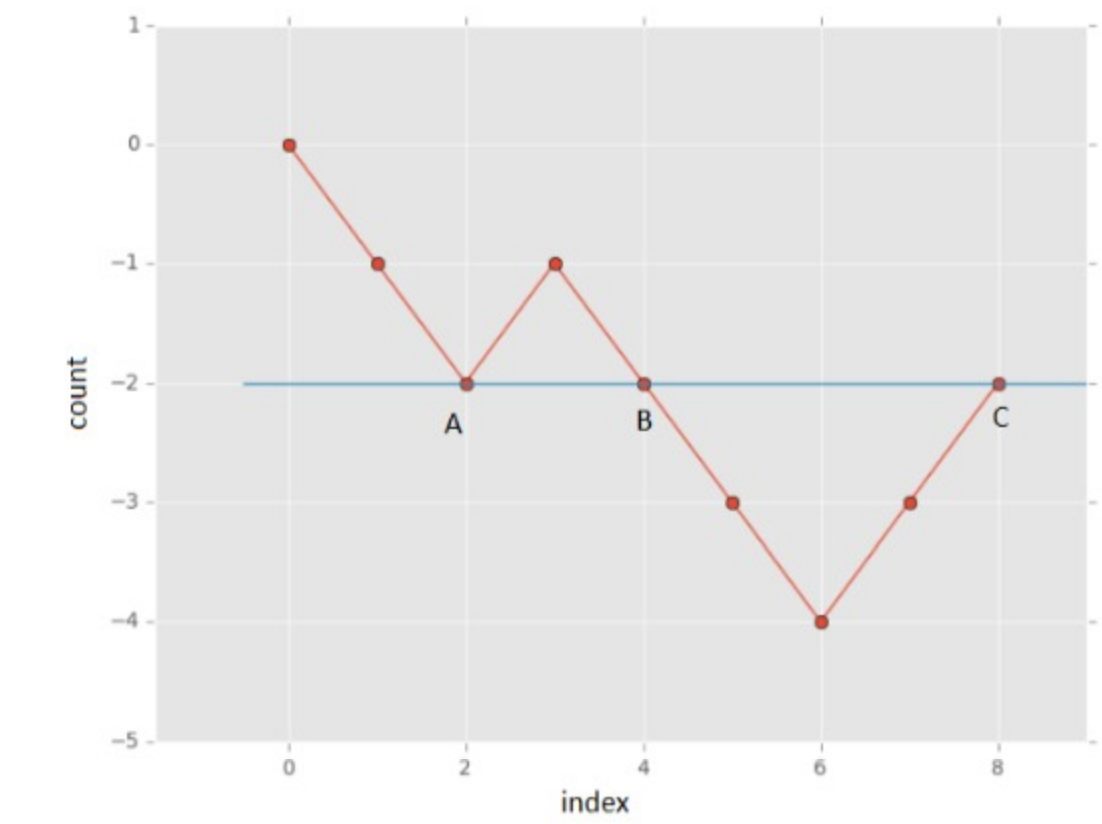
- Time complexity: $O(n^2)$. We consider every possible subarray by traversing over the complete array for every start point possible.
- Space complexity: $O(1)$. Only two variables `zeroes` and `ones` are required.

Approach #2 Using Extra Array [Accepted]

Algorithm

In this approach, we make use of a `count` variable, which is used to store the relative number of ones and zeros encountered so far while traversing the array. The `count` variable is incremented by one for every 1 encountered and the same is decremented by one for every 0 encountered.

We start traversing the array from the beginning. If at any moment, the `count` becomes zero, it implies that we've encountered equal number of zeros and ones from the beginning till the current index of the array(*i*). Not only this, another point to be noted is that if we encounter the same `count` twice while traversing the array, it means that the number of zeros and ones are equal between the indices corresponding to the equal `count` values. The following figure illustrates the observation for the sequence `[0 0 1 0 0 0 1 1]`:



In the above figure, the subarrays between (A,B), (B,C) and (A,C) (lying between indices corresponding to `count = 2`) have equal number of zeros and ones.

Another point to be noted is that the largest subarray is the one between the points (A, C). Thus, if we keep a track of the indices corresponding to the same `count` values that lie farthest apart, we can determine the size of the largest subarray with equal no. of zeros and ones easily.

Now, the `count` values can range between $-n$ to $+n$, with the extreme points corresponding to the complete array being filled with all 0's and all 1's respectively. Thus, we make use of an array `arr`(of size $2n+1$) to keep a track of the various `count`'s encountered so far. We make an entry containing the current element's index (*i*) in the `arr` for a new `count` encountered everytime. Whenever, we come across the same `count` value later while traversing the array, we determine the length of the subarray lying between the indices corresponding to the same `count` values.

JavaCopy

```
1
2 public class Solution {
3
4     public int findMaxLength(int[] nums) {
5         int[] arr = new int[2 * nums.length + 1];
6         Arrays.fill(arr, -1);
7         arr[nums.length] = -1;
8         int maxlen = 0, count = 0;
9         for (int i = 0; i < nums.length; i++) {
10             count = count + (nums[i] == 0 ? -1 : 1);
11             if (arr[count + nums.length] == -1) {
12                 maxlen = Math.max(maxlen, i - arr[count + nums.length]);
13             } else {
14                 arr[count + nums.length] = i;
15             }
16         }
17         return maxlen;
18     }
19 }
20
21
```

Complexity Analysis

- Time complexity: $O(n)$. The complete array is traversed only once.
- Space complexity: $O(n)$. `arr` array of size $2n+1$ is used.

Approach #3 Using HashMap [Accepted]

Algorithm

This approach relies on the same premise as the previous approach. But, we need not use an array of size $2n+1$, since it isn't necessary that we'll encounter all the `count` values possible. Thus, we make use of a `HashMap` `map` to store the entries in the form of (`index`, `count`). We make an entry for a `count` in the `map` whenever the `count` is encountered first, and later on use the corresponding index to find the length of the largest subarray with equal no. of zeros and ones when the same `count` is encountered again.

nums

0	1	2	3	4	5	6
0	1	0	0	1	1	0

Count=0Maxlen=0

▶

1 / 8

JavaCopy

```
1
2 public class Solution {
3
4     public int findMaxLength(int[] nums) {
5         Map<Integer, Integer> map = new HashMap<>();
6         map.put(0, -1);
7         int maxlen = 0, count = 0;
8         for (int i = 0; i < nums.length; i++) {
9             count = count + (nums[i] == 1 ? 1 : -1);
10             if (map.containsKey(count)) {
11                 maxlen = Math.max(maxlen, i - map.get(count));
12             } else {
13                 map.put(count, i);
14             }
15         }
16         return maxlen;
17     }
18 }
19
```

Complexity Analysis

- Time complexity: $O(n)$. The entire array is traversed only once.
- Space complexity: $O(n)$. Maximum size of the `HashMap` `map` will be n , if all the elements are either 1 or 0.


Rate this article: ★★★★★

Previous



Next


Comments: 37

Sort By ▾



Type comment here... (Markdown is supported)

 Preview  Post







maverick009★234


July 14, 2019 2:09 PM

Amazing solution 3. However, I think we don't need to initialise map with map.put(0,-1) which is a bit confusing
Count is like prefix sum. If at some point, it becomes 0, it means array from the beginning contains equal number of 0's and 1's.

Read More

69    





SHOW 4 REPLIES




chwlN★180

June 23, 2017 9:50 PM

I think in the count & index table in Solution#3 slides, the first column should be count = 0 indexed at -1. This really confused me for a while..

56    

SHOW 5 REPLIES







jackkchi★14


April 14, 2020 5:38 AM

Approach #3 explanation has a typo mistake. It says:
"Thus, we make use of a HashMap mapmap to store the entries in the form of (index, count)"
But the wording implied by the solution given should say:

Read More

12    




SHOW 2 REPLIES




sjoxavier89★15

April 19, 2020 11:22 AM

I found the explanation of approach 2 very difficult to understand!

11    







miladinho★234

April 15, 2020 2:15 AM


This is a retarded problem, why:

1. Given the next best solution after the brute force one is extremely hard to figure out unless you have some prior insight, the n^2 solution should not be disallowed.
2. There is only one trick to make the algo more efficient, and that is NOT trivial to figure out:

Read More

19    





SHOW 1 REPLY




user4805j★6

April 13, 2020 3:57 PM

Why can't we use stack?

6    

SHOW 1 REPLY








chulman444★5

April 13, 2020 8:45 PM

It confused me at first too. Having `{ 0: -1}` does make as much sense as `maxlen = Math.max(maxlen, i+1)`. It just plays around the fact that the length is based on the first index at which the sum was seen (so that we keep track of the maximum length) and the last index the sum was seen. It makes much more sense when you think about each element in the passed array to be a vector that advances towards the right axis with '0' meaning "go -1 in x-axis" and '1' meaning "go +1 in x-axis"

Read More





5    




Nagarjuna9★10

January 4, 2020 4:02 AM

Can anyone please explain the question? I still don't understand what the question is about? It is asking for contiguous sub array. If that's the case, why does [0,0,1,1,1] equals 6 where as [0,1,1,0,1,1,0] equals 4?

6    





SHOW 3 REPLIES




hungry_coder_n★11

April 13, 2020 11:13 PM

Why did they do Arrays.fill(arr, -2); in in approach-2? Can someone please explain?

3    





SHOW 2 REPLIES



vincentFeng0101★52

April 13, 2020 7:53 PM

it is clear that in Approach 2, the subarrays between (A,B) is not a valid subarray with equal 0 and 1, why author says that?

3    

SHOW 1 REPLY