Find Index of 0 to be replaced with 1 to get longest continuous sequence of 1s in a binary array

Given an array of 0s and 1s, find the position of 0 to be replaced with 1 to get longest continuous sequence of 1s. Expected time complexity is O(n) and auxiliary space is O(1).
Example:

```
Input:
   arr[] =  {1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1}
Output:
  Index 9
Assuming array index starts from 0, replacing 0 with 1 at index 9 causes
the maximum continuous sequence of 1s.

Input:
   arr[] =  {1, 1, 1, 1, 0}
Output:
  Index 4
```

We strongly recommend to minimize the browser and try this yourself first.

A **Simple Solution** is to traverse the array, for every 0, count the number of 1s on both sides of it. Keep track of maximum count for any 0. Finally return index of the 0 with maximum number of 1s around it. The time complexity of this solution is $O(n^2)$.

Using an **Efficient Solution**, the problem can solved in O(n) time. The idea is to keep track of three indexes, current index (*curr*), previous zero index (*prev_zero*) and previous to previous zero index (*prev_prev_zero*). Traverse the array, if current element is 0, calculate the difference between *curr* and *prev_prev_zero* (This difference minus one is the number of 1s around the prev_zero). If the difference between *curr* and *prev_prev_zero* is more than maximum so far, then update the maximum. Finally return index of the prev_zero with maximum difference.

Following are C++ and Java implementations of the above algorithm.

C++

```cpp
// C++ program to find Index of 0 to be replaced with 1 to get
// longest continuous sequence of 1s in a binary array
#include<iostream>
using namespace std;

// Returns index of 0 to be replaced with 1 to get longest
// continuous sequence of 1s.  If there is no 0 in array, then
// it returns -1.
int maxOnesIndex(bool arr[], int n)
{
    int max_count = 0;  // for maximum number of 1 around a zero
    int max_index;  // for storing result
    int prev_zero = -1;  // index of previous zero
    int prev_prev_zero = -1; // index of previous to previous zero

    // Traverse the input array
    for (int curr=0; curr<n; ++curr)
    {
        // If current element is 0, then calculate the difference
        // between curr and prev_prev_zero
        if (arr[curr] == 0)
        {
            // Update result if count of 1s around prev_zero is more
            if (curr - prev_prev_zero > max_count)
            {
                max_count = curr - prev_prev_zero;
                max_index = prev_zero;
            }

            // Update for next iteration
            prev_prev_zero = prev_zero;
            prev_zero = curr;
        }
    }

    // Check for the last encountered zero
    if (n-prev_prev_zero > max_count)
       max_index = prev_zero;

    return max_index;
}

// Driver program
int main()
{
    bool arr[] = {1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Index of 0 to be replaced is "
         << maxOnesIndex(arr, n);
    return 0;
}
```

Run on IDE

Java

```java
// Java program to find Index of 0 to be replaced with 1 to get
// longest continuous sequence of 1s in a binary array

import java.io.*;

class Binary
```

```
{
    // Returns index of 0 to be replaced with 1 to get longest
    // continuous sequence of 1s.  If there is no 0 in array, then
    // it returns -1.
    static int maxOnesIndex(int arr[], int n)
    {
        int max_count = 0;  // for maximum number of 1 around a zero
        int max_index=0;  // for storing result
        int prev_zero = -1;  // index of previous zero
        int prev_prev_zero = -1; // index of previous to previous zero

        // Traverse the input array
        for (int curr=0; curr<n; ++curr)
        {
            // If current element is 0, then calculate the difference
            // between curr and prev_prev_zero
            if (arr[curr] == 0)
            {
                // Update result if count of 1s around prev_zero is more
                if (curr - prev_prev_zero > max_count)
                {
                    max_count = curr - prev_prev_zero;
                    max_index = prev_zero;
                }

                // Update for next iteration
                prev_prev_zero = prev_zero;
                prev_zero = curr;
            }
        }

        // Check for the last encountered zero
        if (n-prev_prev_zero > max_count)
            max_index = prev_zero;

        return max_index;
    }


    // Driver program to test above function
    public static void main(String[] args)
    {
        int arr[] = {1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1};
        int n = arr.length;
        System.out.println("Index of 0 to be replaced is "+
                           maxOnesIndex(arr, n));
    }
}
/* This code is contributed by Devesh Agrawal */
```

Run on IDE

Output:

```
Index of 0 to be replaced is 9
```

Time Complexity: O(n)

Auxiliary Space: O(1)

This article is contributed by **Ankur Singh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

53 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

(Login to Rate and Mark)

3.1 Average Difficulty : **3.1/5.0**
Based on **8** vote(s)

Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Join the discussion…

Varun Joshi • 2 months ago

http://code.geeksforgeeks.org/...

O(n) in C

• Reply • Share ›

Naveen Kumar • 3 months ago

here we should return max_index-1 and for the last zero which will be pointed by curr ,this code

"if (n-prev_prev_zero > max_count) max_index = prev_zero;"

should be

"if (n-prev_zero>max_count) max_index =curr;" because we are checking for last zero which is curr.

• Reply • Share ›

Anurag Singh Naveen Kumar • 22 days ago

No.The given solution is correct. at last curr will be n. so "if (n-prev_zero>max_count) max_index =curr;" then max_index=curr will give max_index=n, which wrong.

• Reply • Share ›

shaidan15 • 4 months ago

I have a better method:

take this serie and devide it on 2 halves :

1, 1, 0, 0, 1, 0, |1|, 1, 1, 0, 1, 1, 1 and then XOR it , it will give:

0,0, 1, 1, 0, 1, |0|, 0 , 0 ,1, 0,0, 0

now the right half has only a UNIQUE one : 1 and the rest are 0z , we choose this half and the index will be

the index of that 1 which is : 9 ! if they werent a UNIQUE 1 we repeat step 1 on the side with less 1.

in example 2 we have :

1, 1, |1|, 1, 0 we XOR it :

0 ,0 , |0|,0,1 we got unique 1 in right side at index : 4

I think this works also if we start with 0 for example , suppose we have 01110 instead of (11110)

0 1 |1| 1 0 xor it gives 1 0 |0| 0 1 now we have 2 uniques 1 one in left and one in right , means both

are candidate indexes.. either index 0 or index 4 makes a longest streak.

so we can implement this using only 2 operations, XOR and (+) we add the halves numbers if the sum equal to one then that half has a UNIQUE 1 !

• Reply • Share ›

Billionaire • 5 months ago

Brilliant solution! Use 3 pointers to track

1 • Reply • Share ›

invincible • 5 months ago

@GeeksforGeeks
in the second last line...i think it should be "n-1-prev_prev_zero"...

• Reply • Share ›

devakar verma • 5 months ago

anyone plz explain i am getting confuesd.

• Reply • Share ›

stellari • 6 months ago

The "Simple Solution" might appear as an O(n^2) solution, but if you think about it, it is actually O(n), because when examine the 1-neighbors of each 0, you NEVER COUNT ANY NUMBER 1 MORE THAN TWICE.
Imagine such a sequence of interleaving 0's and 1's: [0 x M1, 1 x N1, 0 x M2, 1 x N2 , ... 0 X Mk, 1 x Nk], where Mi and Ni represent the number of 0's and 1's, repsectively.
So
1. For every 1, we can simply skip it in O(1) time. Overall, we spend only O(N1 + N2 +... + Nk) time on 1's
2. For every 0 that does not have "1" neighbors, we can also process each of them in O(1) time. So we spend O(M1 + M2 + ... Mk) on those 0's
3. For every 0 that has "1" neighbors: The # of 1-neighbors of a 0 at the beginning of sequence Mi is Ni, and that of the 0 at the end of Mi is N(i+1). So overall, we only spend O((N1 + N2) + (N2 + N3) + ... (Nk-1 + Nk)) = O(N1 + N2 + ... + Nk)
Add 1 to 3 together, we have the overall time complexity of O(n)

• Reply • Share ›

Bewkoof_coder • 6 months ago

@geeks for geeks
simple O(n) time complexty and O(1) space.
Algo is.
1) count the no of ones on left of 0.i.e sum1;
2)count the no of ones on right side of a 0(i.e till next 0 occurs or till last index(sum2).
3)if (sum1 + sum2) > max (initially max=0) then save the index of prev zero as index and now sum1=sum2
now for sum2(right side 1,s of prev 0) bcms sum1(left side 1,s for next 0)and make sum2=0 and start count sum2(right side 1,s from the current 0) till the n(length)

sum2=0 and start count sum2(right side 1's from the current 0)..till the n(length).

code is below..

int index(int *A,int n){

int sum1=0,sum2=0,max=0,j,i=0,index;

while(A[i]!=0){

++i;

see more

1 • Reply • Share ›

Klaus • 7 months ago

http://ideone.com/lSTWx9 Simple Sliding window like approach.

1 • Reply • Share ›

Akshay • 7 months ago

@GeeksforGeeks
Another approach don't know if it is correct or wrong :P

taking example as {1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1}

Initially iterate for consecutive ones keeping a count variable

and convert the array as follows:-
1 2 0 0 1 0 1 1 3 0 1 1 3.
Next now iterate from bckwards and convert array as follows :-
2 2 0 0 1 0 3 3 3 0 3 3 3.
This u can convert taking the element in last index and fill the sum in consecutive ones until u encounter a zero.
When u encounter a zero iterate again until get next sum.Again as soon as u get a sum repeat the copy into consecutive ones and so on.

Next check the zero with index which has left and right as maximum sum. This u can do keeping track of maximum so far.

So total time complexity would:-

O(n)+O(n)+O(n)-----> which is eventually O(n).

Correct me if i am wrong.

1 • Reply • Share ›

Avata This comment was deleted.

Goku Guest • 6 months ago

good solution gaurav :D

• Reply • Share ›

Mr. Lazy Goku • 6 months ago

haha...sorry that sucks ... its no different :P ..deleting it ... thanks for the poke ;)

• Reply • Share ›

Sunil Sharma • 8 months ago

can somebody explain me this .

after encountering first zero at index 2 .
maxcount= curr-prev_prev_zero
i.e
maxcount = 2-(-1)=3
maxindex= 2;

prev_prev_zero= -1.
prev_zero =2.

now for next zero at location 3
condition (curr-prev_prev_zero > max) is true
and we are changing the value of

maxcount to 3-(-1) i.e 4 . but its not true as the maxcount is 2 so far.

please explain this.

• Reply • Share ›

Sunil Sharma Sunil Sharma • 8 months ago

got it

• Reply • Share ›

Lehar Sunil Sharma • 7 months ago

Can you explain it to me? I have the same doubt.

• Reply • Share ›

Nikhil Lehar • 4 months ago

I have the same doubt, maxcount should still be 3 and not 4 which the algorithm would calculate???

• Reply • Share ›

Mohit • 9 months ago

@GeeksForGeeks
Traverse the array, if current element is 0, calculate the difference between curr and prev_prev_zero (This difference minus TWO is the number of 1s around the prev_zero). Since, we do not need the maxcount in this case, we ONLY need the position of zero to be changed to 1 in order to get longest no. of 1's. So, the code is correct according to the question.But, adjacent number of 1's are wrong, which must be curr - prev_prev_zero - 2.

1 • Reply • Share ›

dark_knight • 10 months ago

There can be a different interpretation for the above problem . Lets say we have two types of sum 1) includingZeroSum = consecutive sum when 0 has been included 2) excludingZeroSum = consecutive sum when only 1s have been included . Following is the way to update both types of sum , like if current element is '0' then includingZeroSum = excludingZeroSum +1 and excludingZeroSum =0 . Now keep the global max and index for the last zero element .

https://ideone.com/pVCR8b

• Reply • Share ›

Sumit Kesarwani • a year ago

https://gist.github.com/sumitd...

TC-->O(N) SC-->O(1)

• Reply • Share ›

aastha singh • a year ago

http://ideone.com/uFXjO2

• Reply • Share ›

Avinash Abhi • a year ago

the max_count statement should be
max_count = curr - prev_prev_zero - 1;

2 • Reply • Share ›

Sameer • a year ago

See another logic of doing this program at following link-http://ideone.com/U42PpU

• Reply • Share ›

nishant gupta • a year ago

Check out this :
http://ideone.com/v72Lfe

and let me know your thoughts !!

• Reply • Share ›

Vinod • a year ago

http://ideone.com/oWoyFr

• Reply • Share ›

Alok • a year ago

if (curr - prev_prev_zero > max_count) should be if (curr - prev_prev_zero -1 > max_count) as the former gives one extra 1 count which is wrong

• Reply • Share ›

Tyrion • a year ago

Another approach is to replace all contiguous 1s with their sum. So for array {1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1} sum array would be {2, 2, 0, 0, 1, 0, 3, 3, 3, 0, 3, 3, 3} . Once we have this array find the index of zero whose left and right number sum is maximum.

http://ideone.com/wLhvUF

3 • Reply • Share ›

Sumit Kesarwani ➔ Tyrion • a year ago

i was also thinking the same approche..but its nice..

• Reply • Share ›

Alok ➔ Tyrion • a year ago

This is a nice approach but for each 1 you are searching linearly for all consecutive ones and then adding up which gives worst case time complexity of O(N^2). And you need at least two passes for calculating the required index.

2 • Reply • Share ›

Akshay ➔ Alok • 7 months ago

Initially iterate for consecutive ones keeping a count variable
and convert the array as follows:-
1 2 0 0 1 0 1 1 3 0 1 1 3.
Next now iterate from bckwards and convert array as follows :-
2 2 0 0 1 0 3 3 3 0 3 3 3.
This u can convert taking the element in last index and fill the sum in consecutive ones until u encounter a zero.
When u encounter a zero iterate again until get next sum.Again as soon as u get a sum repeat the copy into consecutive ones and so on.

Next check the zero with index which has left and right as maximum sum. This u can do keeping track of maximum so far.

So total time complexity would:

So total time complexity would:-

O(n)+O(n)+O(n)-----> which is eventually O(n).

Correct me if i am wrong.

1 • Reply • Share ›

Badal • a year ago

My Solution: Pretty simple and takes O(n) time complexity:

```
public int positionOfZero(int[] arr){
int beforePrev = -1;
int prev = -1;
int curr = 0;
int max = 0;
int maxindex = -1;
int count = 0;

for (int i=0; i< arr.length; i++){
if (arr[i]==0){
count++;
if(count > 1){
beforePrev = prev;
prev = curr;
curr = i;
int diff = curr - beforePrev -1;
```

see more

1 • Reply • Share ›

Sumit Kesarwani ➦ Badal • a year ago

greate..

• Reply • Share ›

AKASH SHARMA • a year ago

My algo takes O(n) time complexity with constant space.Both index of zero and number of 1's are calculated.

https://github.com/akash-sharm...

• Reply • Share ›

AMIT PATIAL • a year ago

I have used the logic that the right 1's sequence for the first zero is the left sequence for the next zero (in case of non-consecutive zeros) for the consecutive zeros, I have made the value of leftmost sequence length zero.

The code is given below:

```
int leftSeqLengh = 0,prvRightSeqLength = 0, finalIndex = -1, localIndex = -1, maxOne = 0;

boolean consecutiveZero = false;

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

if (arr[i] == 0) {

localIndex= i;

if (consecutiveZero) {

leftSeqLengh = 0;
```

see more

• Reply • Share ›

rihansh • a year ago

http://ideone.com/orpnPN

A simple implementation of this problem :D

• Reply • Share ›

Tanay • a year ago

Efficient java solution taking O(n) time and O(1) space..
http://ideone.com/E8lmym

• Reply • Share ›

shashi • a year ago

```
/**
* resultIndex stores which index of 0 to replace. and returns number of 1's
* that replacement will make.
*
* References back to Kadane's algo.
*
* @param data
* @return
*/
public int findmax1s(int[] data) {
int max = -1;
int resultIndex = -1;
int left = 0, right = 0;
```

```
for (int countIndex = 0; countIndex < data.length; countIndex++) {
if (data[countIndex] == 0) {
if (max < left + right) {
max = left + right;
resultIndex = countIndex;
```

see more

• Reply • Share ›

Sorabh Hamirwasia • a year ago

I think the solution posted above by Ankur will not give the correct value for "maxCount". That is if we want above program to print the correct count of maximum 1's obtained after replacing the found zero, it will not give right answer.

Here is my version of the code. RunTime = O(n) and Space = O(1)
http://ideone.com/9paG4H

• Reply • Share ›

intuit01 • a year ago

How about
http://ideone.com/4d9FHk

Logic :
Keep counters for left, right, cur_sum, sum

Space Complexity : O(1)
Time Complexity : O(n)

• Reply • Share ›

Nikhil Agarwal • a year ago

What if the input is

a) {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 } - Ans can be 0,2,13

b) {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 } - Ans should list both 1 and 11

• Reply • Share ›

intuit01 ➚ Nikhil Agarwal • a year ago

http://ideone.com/4d9FHk

In a.) answer can 0, 2 or 12
My solution gets the last 1

• Reply • Share ›

rationalcoder • a year ago

```
int prevZeroIndex = 0;
int lengthBetweenZeros = 0;
int[] arr = { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0 };
int outputindex = 0;
for (int currentIndex = 0; currentIndex < arr.Length - 1; currentIndex++)
{
if (arr[currentIndex] == 0)
{
if ((currentIndex - prevZeroIndex) >= lengthBetweenZeros)
{
lengthBetweenZeros = (currentIndex - prevZeroIndex);
outputindex = currentIndex;
}
prevZeroIndex = currentIndex;
}
}
Console.WriteLine("The output index is " + outputindex);
Console.ReadLine();
```

1 • Reply • Share ›

Eric • a year ago

How about traversal from left to right, and right to left? count continuous 1s for each 0 both left and right. 0 with 0 neighbor gets 0

• Reply • Share ›

kafee786 ➔ Eric • a year ago

Then it will take extra O(n) memory space for storing the count.

• Reply • Share ›

Amazon Aspirant • a year ago

here comes the most awaited interview question .....

2 • Reply • Share ›

Amazon Aspirant ➔ Amazon Aspirant • a year ago

Kadanes modification

1 • Reply • Share ›

Sumit Kesarwani ➔ Amazon Aspirant • a year ago

```
private int getIndex0s(int[] a)
{
try
{
int max_so_far=Integer.MIN_VALUE;
int max_end_here=0;
```

int max_end_here=0;
int ind_0s = 0;
for(int i=0;i<a.length;i++) {="" *="" {1,="" 1,="" 0,="" 0,="" 1,="" 0,="" 1,="" 1,="" 1,="" 0,="" 1,="" 1,="" 1}*="" max_end_here="max_end_here+a[i];" if(max_end_here<="0)" {="" max_end_here="0;" }="" if(a[i]="=0&&max_end_here">=0&&max_so_far<max_end_here) {="" max_so_far="max_end_here;" ind_0s="i;" }="" }="" return="" ind_0s;="" }="" catch(exception="" ex)="" {="" ex.printstacktrace();="" }="" return="" -1;="" }="">

▲ | ▼ • Reply • Share ›

Amazon Aspirant ➚ Sumit Kesarwani • a year ago

Can u plz write the code n paste the ideone link instead

▲ | ▼ • Reply • Share ›

Sumit Kesarwani ➚ Amazon Aspirant • a year ago

?

▲ | ▼ • Reply • Share ›

Load more comments