Longest Span with same Sum in two Binary arrays

Given two binary arrays arr1[] and arr2[] of same size n. Find length of the longest common span (i, j) where j >= i such that arr1[i] + arr1[i+1] + …. + arr1[j] = arr2[i] + arr2[i+1] + …. + arr2[j].

Expected time complexity is Θ(n).

Examples:

```
Input: arr1[] = {0, 1, 0, 0, 0, 0};
       arr2[] = {1, 0, 1, 0, 0, 1};
Output: 4
The longest span with same sum is from index 1 to 4.

Input: arr1[] = {0, 1, 0, 1, 1, 1, 1};
       arr2[] = {1, 1, 1, 1, 1, 0, 1};
Output: 6
The longest span with same sum is from index 1 to 6.

Input: arr1[] = {0, 0, 0};
       arr2[] = {1, 1, 1};
Output: 0

Input: arr1[] = {0, 0, 1, 0};
       arr2[] = {1, 1, 1, 1};
Output: 1
```

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

Method 1 (Simple Solution)

One by one by consider same subarrays of both arrays. For all subarrays, compute sums and if sums are same and current length is more than max length, then update max length. Below is C++ implementation of simple approach.

```
// A Simple C++ program to find longest common
// subarray of two binary arrays with same sum
#include<bits/stdc++.h>
using namespace std;
```

```
// Returns length of the longest common subarray
// with same sum
int longestCommonSum(bool arr1[], bool arr2[], int n)
{
    // Initialize result
    int maxLen = 0;

    // One by one pick all possible starting points
    // of subarrays
    for (int i=0; i<n; i++)
    {
       // Initialize sums of current subarrays
       int sum1 = 0, sum2 = 0;

       // Conider all points for starting with arr[i]
       for (int j=i; j<n; j++)
       {
           // Update sums
           sum1 += arr1[j];
           sum2 += arr2[j];

           // If sums are same and current length is
           // more than maxLen, update maxLen
           if (sum1 == sum2)
           {
             int len = j-i+1;
             if (len > maxLen)
                maxLen = len;
           }
       }
    }
    return maxLen;
}

// Driver progra+m to test above function
int main()
{
    bool  arr1[] = {0, 1, 0, 1, 1, 1, 1};
    bool  arr2[] = {1, 1, 1, 1, 1, 0, 1};
    int n = sizeof(arr1)/sizeof(arr1[0]);
    cout << "Length of the longest common span with same "
            "sum is "<< longestCommonSum(arr1, arr2, n);
    return 0;
}
```

Run on IDE

Output:

```
Length of the longest common span with same sum is 6
```

Time Complexity: $O(n^2)$

Auxiliary Space: O(1)

Method 2 (Using Auxiliary Array)

The idea is based on below observations.

1. Since there are total n elements, maximum sum is n for both arrays.
2. Difference between two sums varies from **-n** to **n**. So there are total 2n + 1 possible values of difference.
3. If differences between prefix sums of two arrays become same at two points, then subarrays between these two points have same sum.

Below is Complete Algorithm.

1. Create an auxiliary array of size 2n+1 to store starting points of all possible values of differences (Note that possible values of differences vary from -n to n, i.e., there are total 2n+1 possible values)
2. Initialize starting points of all differences as -1.
3. Initialize **maxLen** as 0 and prefix sums of both arrays as 0, **preSum1** = 0, **preSum2** = 0
4. Travers both arrays from i = 0 to n-1.
    a. Update prefix sums: preSum1 += arr1[i], preSum2 += arr2[i]
    b. Compute difference of current prefix sums: **curr_diff** = preSum1 – preSum2
    c. Find index in diff array: **diffIndex** = n + curr_diff // curr_diff can be negative and can go till -n
    d. **If** curr_diff is 0, then i+1 is maxLen so far
    e. **Else If** curr_diff is seen first time, i.e., starting point of current diff is -1, then update starting point as i
    f. **Else** (curr_diff is NOT seen first time), then consider i as ending point and find length of current same sum span. If this length is more, then update maxLen
5. Return maxLen

Below is C++ implementation of above algorithm.

```
// A O(n) and O(n) extra space C++ program to find
// longest common subarray of two binary arrays with
// same sum
#include<bits/stdc++.h>
using namespace std;

// Returns length of the longest common sum in arr1[]
// and arr2[]. Both are of same size n.
int longestCommonSum(bool arr1[], bool arr2[], int n)
{
    // Initialize result
    int maxLen = 0;

    // Initialize prefix sums of two arrays
    int preSum1 = 0, preSum2 = 0;

    // Create an array to store staring and ending
    // indexes of all possible diff values. diff[i]
    // would store starting and ending points for
    // difference "i-n"
    int diff[2*n+1];

    // Initialize all starting and ending values as -1.
    memset(diff, -1, sizeof(diff));

    // Traverse both arrays
    for (int i=0; i<n; i++)
    {
        // Update prefix sums
```

```
        preSum1 += arr1[i];
        preSum2 += arr2[i];

        // Comput current diff and index to be used
        // in diff array. Note that diff can be negative
        // and can have minimum value as -1.
        int curr_diff = preSum1 - preSum2;
        int diffIndex = n + curr_diff;

        // If current diff is 0, then there are same number
        // of 1's so far in both arrays, i.e., (i+1) is
        // maximum length.
        if (curr_diff == 0)
            maxLen = i+1;

        // If current diff is seen first time, then update
        // starting index of diff.
        else if ( diff[diffIndex] == -1)
            diff[diffIndex] = i;

        // Current diff is already seen
        else
        {
            // Find lenght of this same sum common span
            int len = i - diff[diffIndex];

            // Update max len if needed
            if (len > maxLen)
                maxLen = len;
        }
    }
    return maxLen;
}

// Driver progra+m to test above function
int main()
{
    bool  arr1[] = {0, 1, 0, 1, 1, 1, 1};
    bool  arr2[] = {1, 1, 1, 1, 1, 0, 1};
    int n = sizeof(arr1)/sizeof(arr1[0]);
    cout << "Length of the longest common span with same "
            "sum is "<< longestCommonSum(arr1, arr2, n);
    return 0;
}
```

Run on IDE

Output:

```
Length of the longest common span with same sum is 6
```

Time Complexity: Θ(n)

Auxiliary Space: Θ(n)

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

11 Comments Category: Arrays

Related Posts:

- 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
- Convert array into Zig-Zag fashion

(Login to Rate and Mark)

3.5 Average Difficulty : **3.5/5.0**
Based on **11** 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.

Chetan • an hour ago

```
// ccc.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "stdio.h"
#include "memory.h"
#define MAX_SIZE 100
int _tmain(int argc, _TCHAR* argv[])
{
freopen( "output.txt", "w", stdout );
freopen( "input.txt", "r", stdin );

int arr1[MAX_SIZE], arr2[MAX_SIZE];
int tc_no=0,i=0,j=0;
int arr_size=0;
int arr1_count=0, arr2_count=0;
int result=0;

scanf("%d",&tc_no);
```

see more

• Reply • Share ›

Chetan • an hour ago

my logic

Since both array are of same size and has 0 and 1 only

1) count number of 1's in both array

2)
if(arr1_count < arr2_count)

{

result = arr_size + arr1_count -arr2_count;

}

else

{

result = arr_size + arr2_count -arr1_count;

}

• Reply • Share ›

Akshit Poddar • 4 days ago

I think the solution can be explained more effectively and mathematically in this fashion.

Let prefixSum1(i) be the prefix sum for the first array, till the i-th position.

Ans similarly prefixSum2(i) for the second array.

Now,

What we seek is:

prefixSum1(i) - prefixSum1 (j) = prefixSum2 (i) - prefixSum2 (j)

This is what the question wants us to find (the longest span, we'll come to this later).

Now, rearranging this:

prefixSum1(i) - prefixSum2 (i) = prefixSum1 (j) - prefixSum2 (j)

Now, if you see the expression is similar on both sides, just with different indexes,
So with every index we can associate a value:
value = prefixSum1(i) - prefixSum2(i)

Now say, if we find another index where this value is repeated.
Aha! we can rearrange to get back the original equation.

Now, finding the longest span is easy.

1 • Reply • Share ›

Harinath Boyapalli ➦ Akshit Poddar • 4 days ago

neatly explained thanks.

• Reply • Share ›

gopal mishra • 8 days ago

c++ implementation for not just binary arrays but all arrays in O(n) & O(n)
http://code.geeksforgeeks.org/...

• Reply • Share ›

stack26 • 9 days ago

Can any body explain the answer with example.Not very easy to understand.What is the motive of taking 2n+1 size array and what is the purpose of diffindex??

• Reply • Share ›

Vishwa Mohan • 15 days ago

Java Implementation :

---

public class LongestSpanSum {

public static void longestLength(int[] a1 , int [] a2){

int[] arr = new int[a1.length];

for(int i=1 ;i<a1.length;i++){ a1[i]="a1[i-1]+a1[i];" a2[i]="a2[i-1]+a2[i];" }="" for(int="" i="0" ;i<a1.length="" ;i++){="" arr[i]="a1[i]-a2[i];" }="" int="" i="0" ;="" int="" j="arr.length" -1="" ;="" while(i<j){="" if(arr[i]="=arr[j]){" system.out.println("length="" :"+="" (j-i)="" +"="" indexes="" :="" "+="" i="" +"="" "+="" j);="" return="" ;="" }else="" if(arr[i+1]="=arr[j]){" system.out.println("length="" :"+="" (j-i)="" +"="" indexes="" :="" "+="" i="" +"="" "+="" j);="" return="" ;="" }else="" if="" (arr[i]="=" arr[j-1]){="" system.out.println("length="" :"+="" (j-i)="" +"="" indexes="" :="" "+="" i="" +"="" "+="" j);="" return="" ;="" }else{="" i++;="" j--;="" }="" }system.out.println("length="" is="" 0="" ");="" }="" public="" static="" void="" main(string="" args[]){="" int="" arr1[]="{0,0,0};" int="" arr2[]="{1," 1,="" 1};="" longestlength(arr1,="" arr2);="" }="" }="">

• Reply • Share ›

Anindya Srivastava ➚ Vishwa Mohan • 13 days ago

Hi Vishwa Mohan,

Thank you for the Java implementation.
It'd be even more awesome if instead writing the code in comments you could please use the GfG IDE: http://code.geeksforgeeks.org/ to share the code.

You can simply paste the code in the IDE. Generate a URL for the same by clicking the button on top right. And, share the URL here.

1 • Reply • Share ›

MOHANA BHATTACHARYA • 15 days ago

can we have a sliding window approach to this?

• Reply • Share ›

Frank Chang ➚ MOHANA BHATTACHARYA • 7 days ago

Hi Sumit Gupta, I found your Big - O(N) solution to take quadratic running time,. Please explain why it does not quadratic running time.

• Reply • Share ›

Frank Chang ➚ Frank Chang • 7 days ago

Hi Sumit, I just verified that your Method 2 solution is Big O(N) time and

space. I sincerely apologize for my hastiness. It will not happen again.

• Reply • Share ›