() () (b)

Average Rating: 4.02 (55 votes)

Copy Copy

Сору

Copy Copy

Copy Copy

Next **1**

209. Minimum Size Subarray Sum 💆 June 5, 2017 | 103.1K views

Given an array of **n** positive integers and a positive integer **s**, find the minimal length of a **contiguous** subarray of which the sum \geq s. If there isn't one, return 0 instead.

Example:

```
Input: s = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: the subarray [4,3]
has the minimal length under the problem constraint.
```

If you have figured out the O(n) solution, try coding another solution of which the time complexity is $O(n \log n)$ n).

Solution

Do as directed in question. Find the sum for all the possible subarrays and update the ans as and when we

Approach #1 Brute force [Time Limit Exceeded]

Intuition

 Initialize ans = INT_MAX Iterate the array from left to right using i:

• Iterate from the current element to the end of vector using j:

- Find the sum of elements from index i to j If sum is greater then s:

 - THE HITHDUDAL PROJECTION S, VECCOI STREET HAMS,
- for (int i = 0; i < n; i++) { for (int j = i; j < n; j++) { 7 int sum = 0; 8 for (int k = i; k <= j; k++) {

```
9
                    sum += nums[k];
 10
 11
               if (sum >= s) {
 12
                     ans = min(ans, (j - i + 1));
 13
                    break; //Found the smallest subarray with sum>=s starting with index i, hence move to next
      index
 14
 15
 16
         }
 17
         return (ans != INT_MAX) ? ans : 0;
 18 }
Complexity Analysis

    Time complexity: O(n<sup>3</sup>).

        \circ For each element of array, we find all the subarrays starting from that index which is O(n^2).
        \circ Time complexity to find the sum of each subarray is O(n).
        \circ Thus, the total time complexity : O(n^2*n) = O(n^3)
```

Intuition

easily find the sum in O(1) time by storing the cumulative sum from the beginning(Memoization). After we

 The only difference is in the way of finding the sum of subarrays: Create a vector sums of size of nums \circ Initialize sums[0] = nums[0] Iterate over the sums vector:

In Approach #1, you may notice that the sum is calculated for every surarray in O(n) time. But, we could

11

sums[i] = sums[i - 1] + nums[i];

for (int i = 0; i < n; i++) {

• Update sums[i] = sums[i-1] + nums[i]

int ans = INT_MAX; vector<int> sums(n); sums[0] = nums[0];for (int i = 1; i < n; i++)

```
12
             for (int j = i; j < n; j++) {
 13
                 int sum = sums[j] - sums[i] + nums[i];
 14
                 if (sum >= s) {
 15
                     ans = min(ans, (j - i + 1));
 16
                     break; //Found the smallest subarray with sum>=s starting with index i, hence move to next
      index
 17
 18
 19
  20
          return (ans != INT_MAX) ? ans : 0;
Complexity analysis

    Time complexity: O(n<sup>2</sup>).

        • Time complexity to find all the subarrays is O(n^2).
        \circ Sum of the subarrays is calculated in O(1) time.
        • Thus, the total time complexity: O(n^2 * 1) = O(n^2)
   • Space complexity: O(n) extra space.
```

We could further improve the Approach #2 using the binary search. Notice that we find the subarray with sum >= s starting with an index i in O(n) time. But, we could reduce the time to $O(\log(n))$ using binary

• Iterate from i=1 to n:

Intuition

C++

9

10

11

Algorithm

5

6

13 14

15 }

O Previous

//ans so on...

for (int i = 1; i <= n; i++)

• Create vector sums of size n+1 with: sums[0] = 0, sums[i] = sums[i-1] + nums[i-1]

greater than s, that is: $to_find = s + sums[i-1]$ Find the index in sums such that value at that index is not lower than the to_find value, say bound

If we find the to_find in sums, then:

1 int minSubArrayLen(int s, vector<int>& nums)

//sums[1]=A[0] : Sum of first 1 elements

- int n = nums.size(); if (n == 0) return 0; int ans = INT_MAX;
- 21 } Complexity analysis Time complexity: O(n log(n)). o For each element in the vector, find the subarray starting from that index, and having sum greater than s using binary search. Hence, the time required is O(n) for iteration over the vector and $O(\log(n))$ for finding the subarray for each index using binary search. • Therefore, total time complexity = $O(n * \log(n))$ • Space complexity: O(n). Additional O(n) space for sums vector Approach #4 Using 2 pointers [Accepted] Intuition Until now, we have kept the starting index of subarray fixed, and found the last position. Instead, we could move the starting index of the current subarray as soon as we know that no better could be done with this index as the starting index. We could keep 2 pointer, one for the start and another for the end of the current subarray, and make optimal moves so as to keep the sum greater than s as well as maintain the lowest size

C++ 1 int minSubArrayLen(int s, vector<int>& nums) 2 {

int n = nums.size(); int ans = INT_MAX;

int left = 0; int sum = 0;

• Initialize left pointer to 0 and sum to 0

While sum is greater than or equal to s:

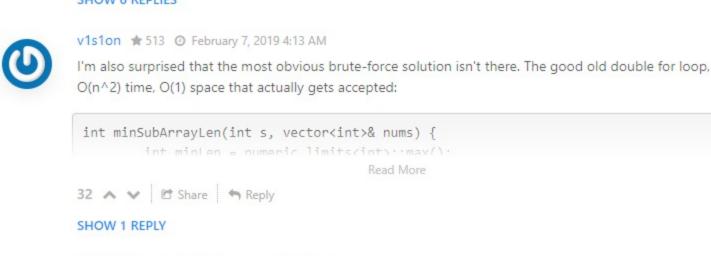
• Add nums i to sum

Iterate over the nums:

Complexity analysis • Time complexity: O(n). Single iteration of O(n). Each element can be visited atmost twice, once by the right pointer(i) and (atmost) once by the

left pointer.

Rate this article: * * * * *

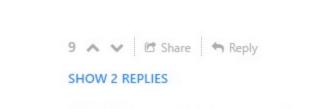


Wrecked * 55 @ January 28, 2019 4:36 AM

public int MinSubArrayLen(int s, int[] nums) {

correct way to use two pointers is

solution 4 is buggy.



5 A V Share Share Reply

kekko1285 🛊 8 ② April 8, 2018 8:12 PM

Read More 6 A V Share Share SHOW 2 REPLIES javak 🛊 3 🗿 November 6, 2017 3:02 PM

@vats.rahul if you sort the array, you are altering the position of the element of the input, hence you cannot provide a correct answer. You have to find all the possible contiguous elements in the input array, not the minimum amount of elements in the array so that this sum is >= of the required sum.

3 A V Share Reply SHOW 1 REPLY

s961206 * 751 • July 19, 2019 5:42 AM Still don't understand why solution 4 works? Could someone please provide formal explanation? 1 A V Share Reply SHOW 3 REPLIES fuyaoli # 95 @ July 21, 2018 10:14 PM

input array, the ans will be INT_MAX at last and it should return INT_MAX since it is a available answer. However, in the given solution, we will return 0. It's wrong. 1 A V C Share Share adheeth_s # 17 @ April 9, 2019 5:32 AM The question by itself is wrong. What is the solution for

Follow up:

get a better subarray that fulfill the requirements (sum \geq s). Algorithm

• Update ans = $\min(ans, (j - i + 1))$ • Start the next ith iteration, since, we got the smallest subarray with $\mathrm{sum} \geq s$ starting from the current index. C++

2 { int n = nums.size(); int ans = INT_MAX;

Space complexity: O(1) extra space.

have stored the cumulative sum in sums, we could easily find the sum of any subarray from i to j. Algorithm The algorithm is similar to Approach #1.

Approach #2 A better brute force [Accepted]

 \circ Sum of subarray from i to j is calculated as: sum = sums[j] - sums[i] + nums[i], , wherein $\operatorname{sums}[j] - \operatorname{sums}[i]$ is the sum from (i+1)th element to the jth element.

1 int minSubArrayLen(int s, vector<int>& nums) int n = nums.size(); if (n == 0) return 0;

• Additional O(n) space for sums vector than in Approach #1. Approach #3 Using Binary search [Accepted]

search. Note that in Approach #2, we search for subarray starting with index i, until we find sum =

 $\operatorname{sums}[j] - \operatorname{sums}[i] + \operatorname{nums}[i]$ that is greater than s. So, instead of iterating linearly to find the sum, we

could use binary search to find the index that is not lower than s - sums[i] in the sums, which can be done

 \circ Find the value $\mathbf{to_find}$ in \mathbf{sum} required for minimum subarray starting from index i to have sum

using lower_bound function in C++ STL or could be implemented manually. Algorithm

• Size of current subarray is given by: bound - (sums.begin() +i-1) Compare ans with the current subarray size and store minimum in ans

vector<int> sums(n + 1, 0); //size = n+1 for easier calculations //sums[0]=0 : Meaning that it is the sum of first 0 elements

12 sums[i] = sums[i - 1] + nums[i - 1];13 for (int i = 1; i <= n; i++) { 14 int to_find = s + sums[i - 1]; auto bound = lower_bound(sums.begin(), sums.end(), to_find); 15 if (bound != sums.end()) { 16 17 ans = min(ans, static_cast<int>(bound - (sums.begin() + i - 1))); 18 19 } 20 return (ans != INT_MAX) ? ans : 0;

possible.

• Update ans = $\min(ans, i + 1 - left)$, where i + 1 - left is the size of current subarray

It means that the first index can safely be incremented, since, the minimum subarray

starting with this index with $\mathrm{sum} \geq s$ has been achieved

Subtract nums left from sum and increment left

7 for (int i = 0; i < n; i++) { sum += nums[i]; while (sum >= s) { 9 10 ans = min(ans, i + 1 - left);11 sum -= nums[left++];

return (ans != INT_MAX) ? ans : 0;

Comments: 47 Sort By -Type comment here... (Markdown is supported) Preview Post tiddlyz # 292 @ August 26, 2018 3:27 PM 1,2 are terrible solutions. Basically brute force solution is made unnecessarily complicate. The simple idea is just to calculate for each i when the sum can be larger or equal to s. No need 3 for loops, 2 loop is enough. Read More 64 A V C Share Reply **SHOW 6 REPLIES**

• Space complexity: O(1) extra space. Only constant space required for left, sum, and i.

Read More 13 A V C Share Reply **SHOW 8 REPLIES** bykov # 16 @ August 10, 2017 2:35 AM 11 [1,2,3,4,5] expected 3 Read More

input 11 [1,2,3,4,5] expected 3

int sum = 0;

SHOW 2 REPLIES

(12345)

SHOW 2 REPLIES

int n = nums.length; int res = Integer.MAX_VALUE; int 1 = 0: Read More

I think there might be problem in the second brute force approach. If there are 2147483647 "1"s in the

[2,7,10]0 or 2? Read More 1 A V E Share Share