Leetcode 209.
Minimum Size Subarray Sum

2024.01.14

원루빈

Problem Description

Given an array of positive integers **nums** and a positive integer **target**, return the **minimal length** of a subarray whose sum is greater than or equal to **target**. If there is no such subarray, return 0 instead.

Example 1:

> **Input:** target = 7, nums = [2,3,1,2,4,3]
> **Output:** 2
> **Explanation:** The subarray [4,3] has the minimal length under the problem constraint.

Example 2:

> **Input:** target = 4, nums = [1,4,4]
> **Output:** 1

Example 3:

> **Input:** target = 11, nums = [1,1,1,1,1,1,1,1]
> **Output:** 0

target = 7, nums = [2,3,1,2,4,3]
output: 2 // [4,3]

1 Start with every possible subarrays (Brute force solution - nested for loops)

target = 7, nums = [2,3,1,2,4,3]

Step1. [2] [2,3] [2,3,1] [2,3,1,2], [2,3,1,2,4], [2,3,1,2,4,3]

Stop when we find the window that sums up to the target value

But since there could be a shorter subarray starting with 3!

1 Start with every possible subarrays (Brute force solution - nested for loops)

target = 7, nums = [2,3,1,2,4,3]

Step 1. [2] [2,3] [2,3,1] [2,3,1,2], [2,3,1,2,4], [2,3,1,2,4,3]

Step 2. [3], [3,1], [3,1,2], [3,1,2,4], [3,1,2,4,3]

… and so on…

→ Thus, we could use nested for loops

Time Complexity: O(n^2), where n is the size of the input array

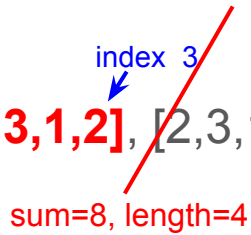
2 Better Approach - Sliding Window Technique

Can we do better than nested for loop?

target = 7, nums = [2,3,1,2,4,3]

Step 1. Starting with 2

[2] [2,3] [2,3,1] **[2,3,1,2]**, [2,3,1,2,4], [2,3,1,2,4,3]



Note that all values are positive!
We know when we start from 3, the sum of the subset would have to be after index 3 since we are now excluding 2.

Step 2. Start with **[3,1,2]**

→ Maintain **two pointers** and shift them accordingly as we identify a valid window.

2 Better Approach - Sliding Window Technique

Can we do better than nested for loop?

target = 7, nums = [2,3,1,2,4,3]

Step 2. Assign left & right pointer

nums = [2,3,1,2,4,3]

Left pointer

Right pointer

CurrentSum= 2

2 Better Approach - Sliding Window Technique

Can we do better than nested for loop?

target = 7, nums = [2,3,1,2,4,3]

Step 3. Iterate right pointer

nums = [2,3,1,2,4,3]

CurrentSum=5
Since 5 < 7 shift R again…

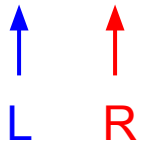
2 Better Approach - Sliding Window Technique

Can we do better than nested for loop?

target = 7, nums = [2,3,1,2,4,3]

Step 3. Iterate right pointer

nums = [2,3,1,2,4,3]



CurrentSum=6

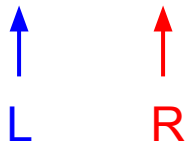
2 Better Approach - Sliding Window Technique

Can we do better than nested for loop?

target = 7, nums = [2,3,1,2,4,3]

Step 4. If target <= CurrentSum, update left pointer

nums = [2,3,1,2,4,3]



CurrentSum=8 ✅
Length of the subarray = 4

Current min Length of the subarray = 4

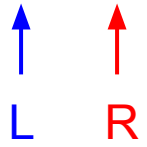
2 Better Approach - Sliding Window Technique

Can we do better than nested for loop?

target = 7, nums = [2,3,1,2,4,3]

Step 5. Repeat which checking current min length of the subarray

nums = [2,3,1,2,4,3]



CurrentSum=8 - 2 = 6

→ Shift the Left pointer

→ CurrentSum -= left value

Current min Length of the subarray = 4

Note

whenever Target <= CurrentSum Left pointer is shifted

2 Better Approach - Sliding Window Technique

Can we do better than nested for loop?

target = 7, nums = [2,3,1,2,4,3]

nums = [2,3,1,2,4,3]

L R

CurrentSum=10 ✅ (but same length)
→ Shift the Left pointer
→ CurrentSum -= left value

Current min Length of the subarray = 4

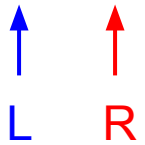
Note
whenever Target <= CurrentSum Left pointer is shifted

2 Better Approach - Sliding Window Technique

Can we do better than nested for loop?

target = 7, nums = [2,3,1,2,4,3]

nums = [2,3,1,2,4,3]



CurrentSum=7 ✅
Length of the subarray = 3 (smaller)

Current min Length of the subarray = 4

2 Better Approach - Sliding Window Technique

Result:

target = 7, nums = [2,3,1,2,4,3]

nums = [2,3,1,2,4,3]

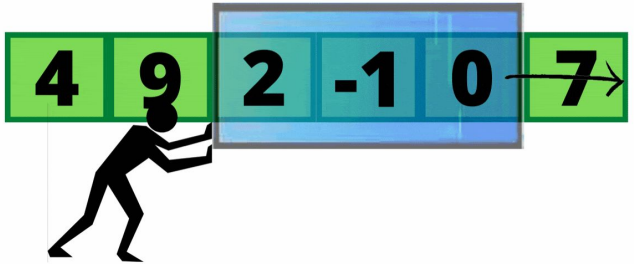
L R

CurrentSum=7 ✅
Length of the subarray = 2 (smaller)

Current min Length of the subarray = 3

Time Complexity: O(n) // running through the array once
Memory Complexity: O(1) // using 2 pointers

3 Sliding Window Technique



[1] Determine Window Size 윈도우 사이즈 정해주기:
Decide on a fixed window size that defines the number of elements to consider at each step.

[2] Initialize and Process 시작점 정해주기:
Start with the initial elements within the window. Perform any initial calculations or operations.

[3] Slide the Window 윈도우를 쭉 한 방향으로 밀어주기:
Iterate through the data, updating the window by adding the next element and removing the leftmost one.

[4] Update and Evaluate:
Adjust calculations or data structures based on the new element.
Evaluate if the current window meets the problem's conditions.
Continue Sliding: Repeat the sliding, updating, and evaluation steps until the end of the data is reached.
Return Result: Return the final result or outcome based on the processed windows.

4 Code Solution

```python
def minSubArrayLen_trial_2(self, target, nums):
    left, right = 0, 0
    total = nums[left]
    cur_result = float('inf')

    while right < len(nums):
        if total >= target:
            cur_result = min(cur_result, abs(left - right) + 1)
            total -= nums[left]
            left += 1
        else:
            right += 1
            if right < len(nums):
                total += nums[right]

    return 0 if cur_result == float('inf') else cur_result
```

감사합니다!

THANK YOU