CS487-Introduction to competitive programming: A2SV

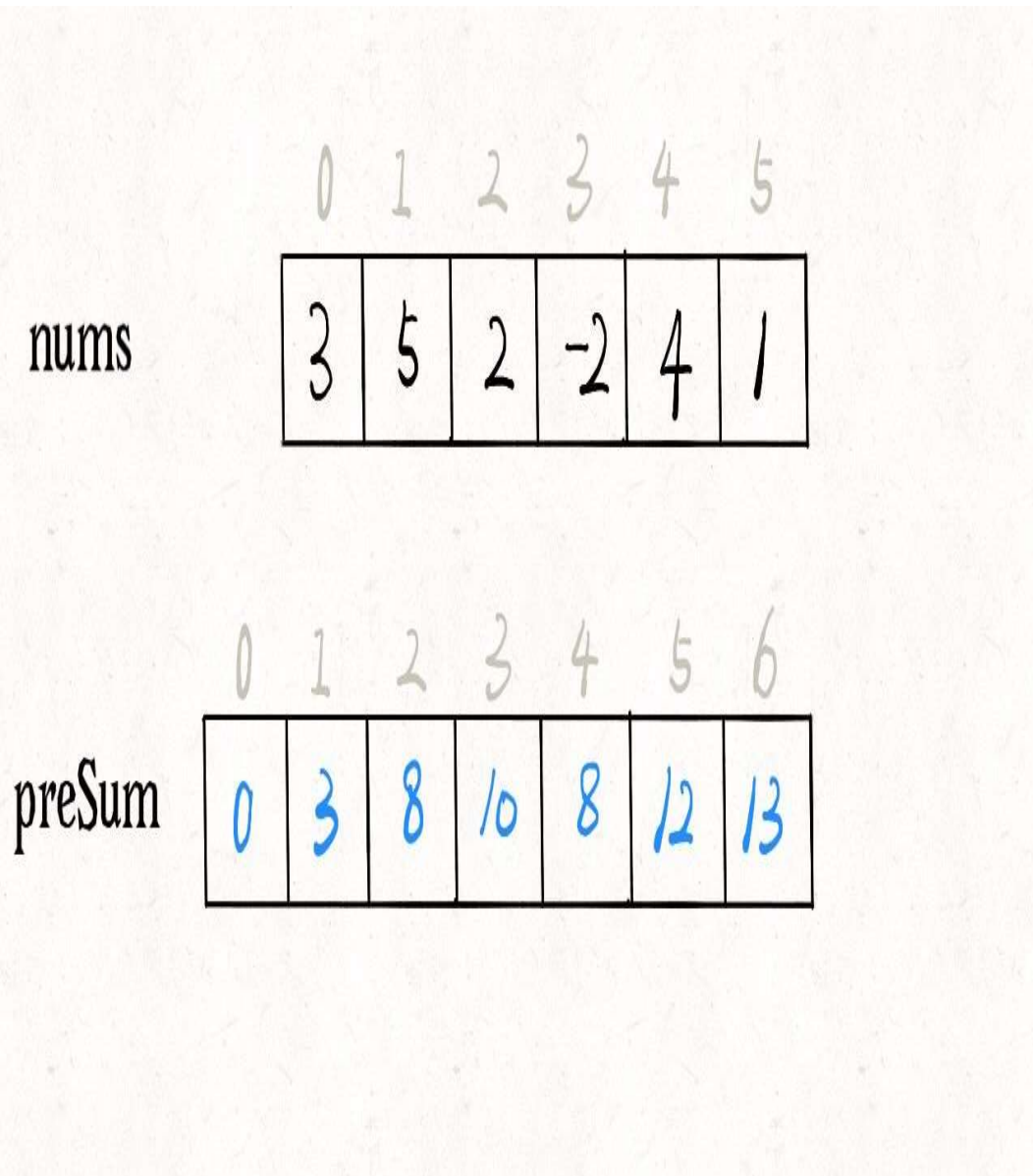
Some popular techniques for problem solving

Introduction

Prefix sum

WHAT IT IS

- for a given sequence, create another sequence to store the sum of prefixes (running totals) of the input sequence
- To calculate the prefix sum,
  - grab the previous value of the prefix sum
  - add the current value of the traversed array.




```
def find_pre_sum(nums):
    pre_sum = [0]
    for i in range(len(nums)):
        pre_sum. append
            (pre_sum[i] + nums[i])
    return pre_sum
```

Exercise 1

Given an array of integers `nums`, you start with an initial **positive** value *startValue*.

In each iteration, you calculate the step by step sum of *startValue* plus elements in `nums` (from left to right).

Return the minimum **positive** value of *startValue* such that the step by step sum is never less than 1.

Example 1:

```
Input: nums = [-3,2,-3,4,2]
Output: 5
Explanation: If you choose startValue = 4, in the third
iteration your step by step sum is less than 1.
                step by step sum
                startValue = 4 | startValue = 5 | nums
                  (4 -3 ) = 1  | (5 -3 ) = 2    |  -3
                  (1 +2 ) = 3  | (2 +2 ) = 4    |   2
                  (3 -3 ) = 0  | (4 -3 ) = 1    |  -3
                  (0 +4 ) = 4  | (1 +4 ) = 5    |   4
                  (4 +2 ) = 6  | (5 +2 ) = 7    |   2
```

Exercise 2

Given a `m x n` matrix `mat` and an integer `k`, return *a matrix* `answer` *where each* `answer[i][j]` *is the sum of all elements* `mat[r][c]` *for*:

- `i - k <= r <= i + k,`
- `j - k <= c <= j + k`, and
- `(r, c)` is a valid position in the matrix.

Example 1:

```
Input: mat = [[1,2,3],[4,5,6],[7,8,9]], k = 1
Output: [[12,21,16],[27,45,33],[24,39,28]]
```

Summary

- Prefix Sum is not hard, yet very useful technique
- However, for more complex problems, simple Prefix Sum technique is not enough.

PREFIX SUM

PROGRAMMING EXERCISES

- Continuous Subarray Sum
- Max Consecutive Ones III
- Maximum Points You Can Obtain from Cards

SLIDING WINDOW

WHAT IT IS

- A very popular technique for performing operations on subarrays or substrings
- Reduce the time complexity from $O(n^3)$ to $O(n^2)$ to $O(n)$

Exercise 1

Given an integer array `nums` and an integer `k`, return `true` if there are two **distinct indices** `i` and `j` in the array such that `nums[i] == nums[j]` and `abs(i - j) <= k`.

Example 1:

```
Input: nums = [1,2,3,1], k = 3
Output: true
```

Example 2:

```
Input: nums = [1,0,1,1], k = 1
Output: true
```

Exercise 2

Given a string `s` consisting only of characters *a*, *b* and *c*.

Return the number of substrings containing **at least** one occurrence of all these characters *a*, *b* and *c*.

Example 1:

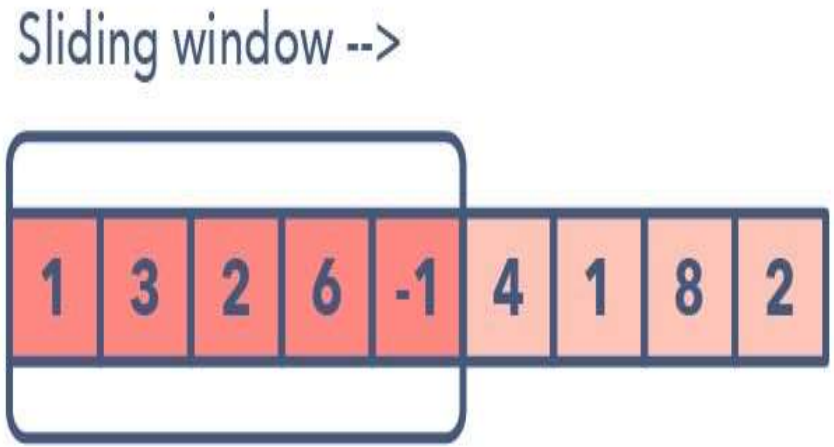
```
Input: s = "abcabc"
Output: 10
Explanation: The substrings containing at least one
occurrence of the characters a, b and c are "abc", "abca",
"abcab", "abcabc", "bca", "bcab", "bcabc", "cab", "cabc" and
"abc" (again).
```

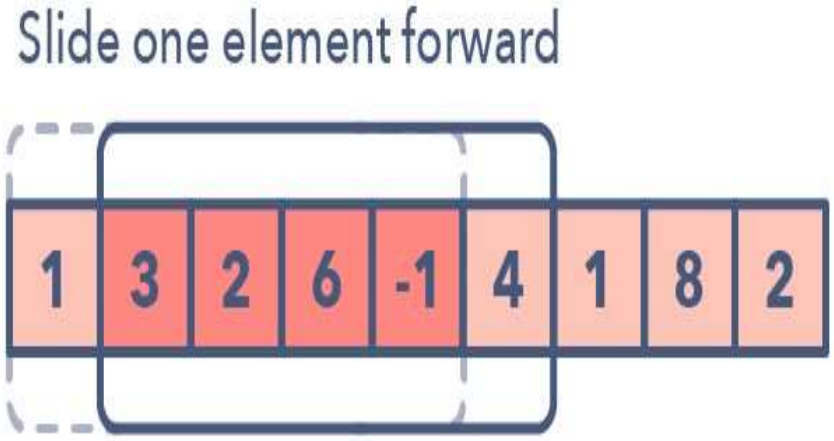
Example 2:

```
Input: s = "aaacb"
Output: 3
Explanation: The substrings containing at least one
occurrence of the characters a, b and c are "aaacb", "aacb"
and "acb".
```

SLIDING WINDOW

RUNNING THROUGH AN EXAMPLE





SLIDING WINDOW

TYPES

There are two types of sliding window:

1. Fixed window length k
2. Dynamic Resizable Window

SLIDING WINDOW

FIXED WINDOW LENGTH K

The length of the window is fixed and it asks you to find something in the window

SLIDING WINDOW

DYNAMIC VARIANTS

Two pointers + criteria: the window size is not fixed, usually it asks you to find the subarray that meets the criteria.

SLIDING WINDOW

VARIANTS OF TWO POINTERS

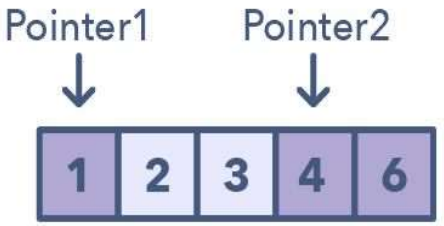
- Opposite Direction: One pointer starts from the beginning while the other pointer starts from the end.

  Example: Two Sum - an input is sorted

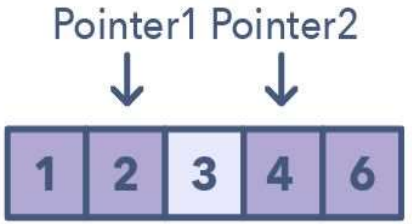
SLIDING WINDOW

RUNNING THROUGH AN EXAMPLE

1 + 6 > target sum, therefore let's decrement Pointer2



1 + 4 < target sum, therefore let's increment Pointer1



2 + 4 == target sum, we have found our pair!

SLIDING WINDOW

VARIANTS OF TWO POINTERS

- Equi-directional: Both start from the beginning, one slow-runner and the other fast-runner.

  Example: Cycle detection on a LinkedList

SLIDING WINDOW

PROGRAMMING EXERCISES

- Max Consecutive Ones III
- Maximum Sum of Two Non-Overlapping Subarrays
- Frequency of the Most Frequent Element

Quote of the day

“It's the possibility of having a dream come true that makes life interesting.”

— Paulo Coelho, The Alchemist