Table of Contents

1. Introduction

Sliding window is a technique used to efficiently process or find patterns in an array or string by maintaining a window of elements that slides through the input. It is especially useful for solving problems that involve finding subarrays or substrings that satisfy certain conditions or have specific properties.

The sliding window technique typically involves using two pointers, often referred to as the "left" and "right" pointers, to define the boundaries of the window. Initially, the window is set to cover a certain number of elements or characters in the input.

As the window slides through the input, the left and right pointers are updated accordingly, either expanding or contracting the window based on certain conditions. The goal is to find the desired pattern or satisfy the given conditions within the window.

2. Algorithm

Here's a basic outline of how the sliding window technique works:

1. Initialize the left and right pointers to define the initial window.
2. While the right pointer is within the bounds of the input:
    - Expand the window by moving the right pointer to the right.
    - Update any necessary data structures or variables to keep track of the window contents or properties.
    - If the current window satisfies the desired condition or pattern:
        - Update the result or perform any required operations.
    - While the window no longer satisfies the condition or pattern:
        - Contract the window by moving the left pointer to the right.

- Update any necessary data structures or variables accordingly.

3. Return the result or perform any final operations.

The sliding window technique can be applied to various problems, such as finding the longest substring with distinct characters, finding a subarray with a certain sum, or finding the minimum window that contains all elements of a given pattern.

By efficiently sliding the window through the input, the sliding window technique often provides an optimized solution with a time complexity of O(n), where n is the size of the input.

3. Example Usage

Let's consider an example problem to illustrate the sliding window technique.

Problem: Given an array of positive integers, find the maximum sum of any contiguous subarray of size k.

Example:

```
def maxSubarraySum(nums, k):
    window_sum = 0
    max_sum = float('-inf')
    left = 0

    for right in range(len(nums)):
        window_sum += nums[right]

        if right >= k - 1:
            max_sum = max(max_sum, window_sum)
            window_sum -= nums[left]
            left += 1

    return max_sum
```

Let's test the function with an example:

```
nums = [2, 3, 1, 5, 1, 2, 6]
k = 3
print(maxSubarraySum(nums, k))  # Output: 8
```

In this example, we are finding the maximum sum of any contiguous subarray of size 3. The sliding window technique is used to efficiently maintain a window of size 3 as it slides through the array.

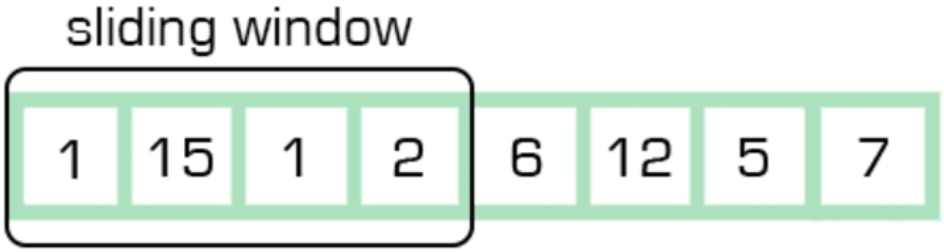
The `left` and `right` pointers are used to define the boundaries of the window.

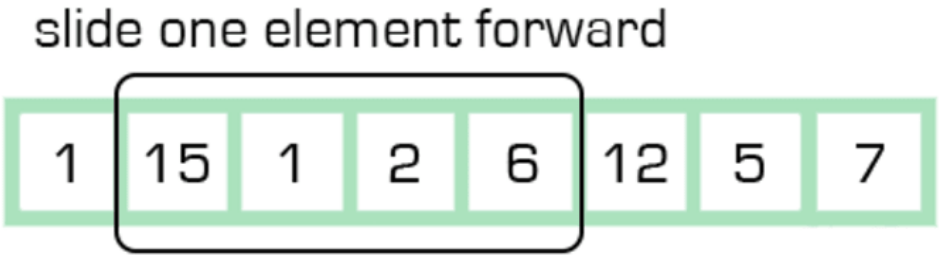
4. Advantages

Advantages of the sliding window technique:

1. Efficiency: The sliding window technique optimizes the solution by reducing unnecessary computations. It avoids re-evaluating subarrays or substrings that have already been processed and focuses only on the elements within the window. This results in improved time complexity, often O(n), where n is the size of the input.
2. Space efficiency: The sliding window technique typically uses a constant amount of extra space, regardless of the size of the input. It avoids creating additional data structures that depend on the input size, leading to better space efficiency.
3. Simplified logic: The sliding window technique often simplifies the problem-solving logic by breaking down complex problems into smaller, more manageable subproblems. It allows you to focus on maintaining the window and updating the necessary variables while sliding through the input.
4. Versatility: The sliding window technique is a versatile approach that can be applied to various problems. It can be used to solve problems related to subarrays, substrings, or sliding windows in general. By understanding and mastering this technique, you can tackle a wide range of algorithmic challenges.

Overall, the sliding window technique offers an efficient and effective approach to solve problems involving patterns or conditions within a sliding window. It combines simplicity, efficiency, and versatility to provide optimized solutions.


sliding window
1
15
1
2
6
12
5
7


slide one element forward
1
15
1
2
6
12
5
7