Data Structures & Algorithms

Sliding Window

B.Bhuvaneswaran, AP (SG) / CSE

9791519152

bhuvaneswaran@rajalakshmi.edu.in

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

Introduction

- Sliding window is another common approach to solving problems related to arrays.
- A sliding window is actually implemented using two pointers!

Subarrays

- Given an array, a **subarray** is a contiguous section of the array.
- All the elements must be adjacent to each other in the original array and in their original order.

Example

- For example, with the array [1, 2, 3, 4], the subarrays (grouped by length) are:
  - [1], [2], [3], [4]
  - [1, 2], [2, 3], [3, 4]
  - [1, 2, 3], [2, 3, 4]
  - [1, 2, 3, 4]

Example

- A subarray can be defined by two indices, the start and end.
- For example, with [1, 2, 3, 4], the subarray [2, 3] has a starting index of 1 and an ending index of 2.
- Let's call the starting index the left bound and the ending index the right bound.
- Another name for subarray in this context is "window".

When should we use sliding window?

- There is a very common group of problems involving subarrays that can be solved efficiently with sliding window.

When should we use sliding window?

- **First**, the problem will either explicitly or implicitly define criteria that make a subarray "valid".
- There are 2 components regarding what makes a subarray valid:
  - A constraint metric. This is some attribute of a subarray. It could be the sum, the number of unique elements, the frequency of a specific element, or any other attribute.
  - A numeric restriction on the constraint metric. This is what the constraint metric should be for a subarray to be considered valid.

Example

- For example, let's say a problem declares a subarray is valid if it has a sum less than or equal to 10.
- The constraint metric here is the sum of the subarray, and the numeric restriction is <= 10.
- A subarray is considered valid if its constraint metric conforms to the numeric restriction, i.e. the sum is less than or equal to 10.

When should we use sliding window?

- **Second**, the problem will ask you to find valid subarrays in some way.
  - The most common task you will see is finding the **best** valid subarray. The problem will define what makes a subarray **better** than another. For example, a problem might ask you to find the **longest** valid subarray.
  - Another common task is finding the number of valid subarrays.

Note

- Whenever a problem description talks about subarrays, you should figure out if sliding window is a good option by analyzing the problem description.
- If you can find the things mentioned above, then it's a good bet.

Example

- Here is a preview of some of the example problems that we will look at in this article, to help you better understand what sliding window problems look like:
  - Find the longest subarray with a sum less than or equal to k
  - Find the longest substring that has at most one "0"
  - Find the number of subarrays that have a product less than k

The algorithm

- The idea behind a sliding window is to consider only valid subarrays.
- Recall that a subarray can be defined by a left bound (the index of the first element) and a right bound (the index of the last element).
- In sliding window, we maintain two variables left and right, which at any given time represent the current subarray under consideration.

The algorithm

- Initially, we have left = right = 0, which means that the first subarray we look at is just the first element of the array on its own.
- We want to expand the size of our "window", and we do that by incrementing right.
- When we increment right, this is like "adding" a new element to our window.

The algorithm

- But what if after adding a new element, the subarray becomes invalid?
  - We need to "remove" some elements from our window until it becomes valid again. To "remove" elements, we can increment left, which shrinks our window.
- As we add and remove elements, we are "sliding" our window along the input from left to right. The window's size is constantly changing - it grows as large as it can until it's invalid, and then it shrinks. However, it always slides along to the right, until we reach the end of the input.

The algorithm

- To explain why this algorithm works, let's look at a specific example. Let's say that we are given a positive integer array nums and an integer k. We need to find the length of the longest subarray that has a sum less than or equal to k. For this example, let nums = [3, 2, 1, 3, 1, 1] and k = 5.

The algorithm

- Initially, we have left = right = 0, so our window is only the first element: [3]. Now, let's expand to the right until the constraint is broken. This will occur when left = 0, right = 2, and our window is: [3, 2, 1]. The sum here is 6, which is greater than k. We must now shrink the window from the left until the constraint is no longer broken. After removing one element, the window becomes valid again: [2, 1].

The algorithm

- Why is it correct to remove this 3 and forget about it for the rest of the algorithm? Because the input only has positive integers, a longer subarray directly equals a larger sum. We know that [3, 2, 1] already results in a sum that is too large. There is no way for us to ever have a valid window again if we keep this 3 because if we were to add any more elements from the right, the sum would only get larger. That's why we can forget about the 3 for the rest of the algorithm.

Length of the longest subarray

- Given an array of positive integers nums and an integer k, find the length of the longest subarray whose sum is less than or equal to k.

Example

- Let's look at an example where nums = [3, 1, 2, 7, 4, 2, 1, 1, 5] and k = 8.
- The longest subarray we found was [4, 2, 1, 1] which means the answer is 4.

Length of the longest substring

- You are given a binary string s (a string containing only "0" and "1").
- You may choose up to one "0" and flip it to a "1".
- What is the length of the longest substring achievable that contains only "1"?
- For example, given s = "1101100111", the answer is 5.
- If you perform the flip at index 2, the string becomes 1111100111.

Subarray Product Less Than K

- Given an array of positive integers nums and an integer k, return the number of subarrays where the product of all the elements in the subarray is strictly less than k.
- For example, given the input nums = [10, 5, 2, 6], k = 100, the answer is 8. The subarrays with products less than k are:
  - [10], [5], [2], [6], [10, 5], [5, 2], [2, 6], [5, 2, 6]

Sum of the subarray

- Given an integer array nums and an integer k, find the sum of the subarray with the largest sum whose length is k.

Closing notes

- Sliding window is extremely common and versatile as a pattern.

Queries?

Thank You…!