Sliding Window collection

Derrick Guo

February 2020

The idea to solve sliding window problem is to use 2 pointers, one on the left and one on the right of the subarray. Slide the right pointer to the right, expanding the window until it violates the requirement stated from the question, then slide the left pointer to shrink the window until the requirement is satisfied again.

1 713. Subarray Product Less Than K

Your are given an array of positive integers nums.

Count and print the number of (contiguous) subarrays where the product of all the elements in the subarray is less than k.

Example 1: Input: nums = [10, 5, 2, 6], k = 100 Output: 8 Explanation: The 8 subarrays that have product less than 100 are: [10], [5], [2], [6], [10, 5], [5, 2], [2, 6], [5, 2, 6]. Note that [10, 5, 2] is not included as the product of 100 is not strictly less than k.

idea:

The requirement of the window is that the product of all elements in the window is less than k. So the way to do this is to expand the window until the products is greater than k, then slide the left pointer to the right to make it less than k again. The number of subarrays in each iteration is right − left + 1, which is the number of subarray ending at index right (starting from index left, moving to the right until index right).

```
class Solution(object):
    def numSubarrayProductLessThanK(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        left = 0
        count = 0
        curprod = 1
        for right in range(len(nums)):
```

```
            if nums[right] >= k:
                curprod = 1
                left = right + 1
                continue
            curprod *= nums[right]
            while curprod >= k:
                curprod /= nums[left]
                left += 1
            count += right - left + 1
        return count
```

2 424. Longest Repeating Character Replacement

Given a string s that consists of only uppercase English letters, you can perform at most k operations on that string.

In one operation, you can choose any character of the string and change it to any other uppercase English character.

Find the length of the longest sub-string containing all repeating letters you can get after performing the above operations.

Note: Both the string's length and k will not exceed 104.

Example 1:

Input: s = "ABAB", k = 2

Output: 4

Explanation: Replace the two 'A's with two 'B's or vice versa.

Example 2:

Input: s = "AABABBA", k = 1

Output: 4

Explanation: Replace the one 'A' in the middle with 'B' and form "AABBBBA". The substring "BBBB" has the longest repeating letters, which is 4.

idea:

If we translate the problem, what it wants us to do is to find the max length of a substring, that the number of major elements + k $<$ its length. So then we can say the requirement of our window is that: right $-$ left $+ 1 -$ major element count $<=$ k. Thus we slide the left pointer to the right until it satisfies this requirement, and in the mean time slide the right pointer to the right as always. At the end of each iteration, we update the max length of the subarray.

```
class Solution(object):
    def characterReplacement(self, s, k):
        """
        :type s: str
        :type k: int
        :rtype: int
```

```
        """
        start = 0
        most_count = 0
        max_len = 0
        counter = collections.defaultdict(int)
        for i, c in enumerate(s):
            counter[c] += 1
            most_count = max(most_count, counter[c])
            while i - start + 1 - most_count > k:
                counter[s[start]] -= 1
                start += 1
            max_len = max(max_len, i-start+1)
        return max_len
```

3 209. Minimum Size Subarray Sum

Given an array of n positive integers and a positive integer s, find the minimal length of a contiguous subarray of which the sum 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.

idea:

There is something different about this question. We can't just slide the left pointer until the sum of the subarray >= s, because sliding the left pointer only decrease the sum. Instead, we update the min length each time when we slide the left, until the sum is no longer >= s.

```
class Solution(object):
    def minSubArrayLen(self, s, nums):
        """
        :type s: int
        :type nums: List[int]
        :rtype: int
        """
        start = 0
        minLen = float('inf')
        curSum = 0
        for end in range(len(nums)):
            curSum += nums[end]
            while curSum >= s:
                minLen = min(minLen, end - start + 1)
                curSum -= nums[start]
                start += 1
        return minLen if minLen < float('inf') else 0
```

```
18
19 # 209
```

4 Fruit into baskets

In a row of trees, the i-th tree produces fruit with type tree[i].

You start at any tree of your choice, then repeatedly perform the following steps:

Add one piece of fruit from this tree to your baskets. If you cannot, stop. Move to the next tree to the right of the current tree. If there is no tree to the right, stop. Note that you do not have any choice after the initial choice of starting tree: you must perform step 1, then step 2, then back to step 1, then step 2, and so on until you stop.

You have two baskets, and each basket can carry any quantity of fruit, but you want each basket to only carry one type of fruit each.

What is the total amount of fruit you can collect with this procedure?

Example 1:

Input: [1,2,1] Output: 3 Explanation: We can collect [1,2,1]. Example 2:

Input: [0,1,2,2] Output: 3 Explanation: We can collect [1,2,2]. If we started at the first tree, we would only collect [0, 1]. Example 3:

Input: [1,2,3,2,2] Output: 4 Explanation: We can collect [2,3,2,2]. If we started at the first tree, we would only collect [1, 2]. Example 4:

Input: [3,3,3,1,2,1,1,2,3,3,4] Output: 5 Explanation: We can collect [1,2,1,1,2]. If we started at the first tree or the eighth tree, we would only collect 4 fruits.

idea:

Let's translate the problem. This description sucks. It simply wants us to find the max length of a subarray that only contains 2 distinct numbers. So we can have 3 place holders, or a hashmap, indicating what the last 2 distinct fruit we saw, and what's the count of the last one. We don't need to track the count of the second last one, because when we see a new fruit, the second last one is thrown away and the last one become the second last one, and the new fruit becomes the last fruit, so the curmax is last fruit count + 1, 1 means the new fruit we just saw. Then we check if we are looking at a fruit that's equal to the last fruit. If so, we increase the last fruit count, otherwise we are looking at a new fruit (or second last fruit, doesn't matter), we set the second last fruit to last fruit, last fruit to new fruit, and last fruit count to 1. then we update maxlen at the end.

```
1 class Solution(object):
2     def totalFruit(self, tree):
3         """
4         :type tree: List[int]
5         :rtype: int
6         """
```

```
        last_fruit = -1
        second_last_fruit = -1
        last_fruit_count = 0
        curmax = 0
        res = 0

        for i, n in enumerate(tree):
            if n == last_fruit or n == second_last_fruit:
                curmax += 1
            else:
                curmax = last_fruit_count + 1

            if n == last_fruit:
                last_fruit_count += 1
            else:
                last_fruit_count = 1
                second_last_fruit = last_fruit
                last_fruit = n

            res = max(curmax, res)
        return res
```