Sliding Window

Mustafa Muhammad

2nd October 2021

1. Maximum Subarray of Size K
2. First Negative Number in every Window of Size k
3. Count Occurences of Anagrams
4. Maximum of all subarrays of size k
5. Variable size Sliding Window — Largest Subarray of sum K
6. Longest Substring with k Unique Characters
7. Longest Substring without Repeating Characters
8. Pick Toys
9. Minimum Window Substring

Identification

1. Question on an array or string
2. Mentions subarray or substring
3. Fixed window size or condition

1 Max Sum SubArray of size K

```
class Solution:
    def maximumSumSubarray (self,K,Arr,N):
        max_sum = 0
        curr = 0
        for i in range(0, len(Arr) -K + 1):
            window = Arr[i:i+K]
            if i == 0:
                curr = sum(window)
            else:
                curr -= Arr[i-1]
                curr += Arr[i+K-1]


            max_sum = max(max_sum, curr)

        return max_sum
```

NOTE : Be careful with calculating the sum

2 First Negative Element in Every Window of Size K

```
def printFirstNegativeInteger( A, N, K):
    res = []
    temp = []
    j = 0
    i = 0
    while(j < len(A)):
        if A[j] < 0:
            temp.append(A[j])

        if j-i+1 < K:
            j+=1

        elif j-i+1 == K:
            if len(temp) != 0:
                res.append(temp[0])
            else:
                res.append(0)

            if A[i]<0:
                temp.pop(0)
            j+=1
            i+= 1

    return res
```

NOTE: BEST TO TRACE THIS SOLUTION OUT, CANT REALLY BE ATTEMPTED USING A FOR LOOP, SINCE THE LAST INCREMENT OF M IS TWICE.

3 Count Occurences of Anagrams

```
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        res = []
        hash_map = {}
        hash_map_p = {}
        for c in p:
            if c in hash_map_p:
                hash_map_p[c] += 1
            else:
                hash_map_p[c] = 1
        i = 0
        j = 0
        while(j < len(s)):
            if s[j] not in hash_map:
                hash_map[s[j]] = 1
            else:
                hash_map[s[j]] += 1
            if j-i+1 < len(p):
                j+=1
            elif j-i+1 == len(p):
                if hash_map == hash_map_p:
                    res.append(i)

                hash_map[s[i]] -= 1
                if hash_map[s[i]] == 0:
                    del hash_map[s[i]]
                i+=1
                j+=1
        return res
```

Brute forcing is to create a window and sort, optimal way is to store previous input using a hashmap.

4 Maximum of all subarrays of size k

```
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        res = []
        i = 0
        j = 0
        ans = []
        queue = []
        if k > len(nums):
            return max(nums)

        while(j < len(nums)):
            while(len(queue) > 0 and queue[-1] < nums[j]):
                queue.pop()
            queue.append(nums[j])
            if j-i+1 < k:
                j+= 1
            elif j-i+1 == k:
                res.append(queue[0])
                if nums[i] == queue[0]:
                    queue.pop(0)
                j += 1
                i += 1
        return res
```

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

Return the max sliding window.

Example 1:

Input: nums = [1,3,-1,-3,5,3,6,7], k = 3

Output: [3,3,5,5,6,7]

Explanation:

Window position Max ——————— —

[1 3 -1] -3 5 3 6 7 3

1 [3 -1 -3] 5 3 6 7 3

1 3 [-1 -3 5] 3 6 7 5

1 3 -1 [-3 5 3] 6 7 5

1 3 -1 -3 [5 3 6] 7 6

1 3 -1 -3 5 [3 6 7] 7

5 Variable Sliding Window