DAY 19

07 December 2022 19:39

Array / String

Fixed size window (window grid)

Computation return

maximum | minimum

subarray | substring

Max Sum Subarray of size K

Basic Accuracy: **49.6%** Submissions: **55120** Points: **1**

Given an array of integers Arr of size **N** and a number **K**. Return the maximum sum of a subarray of size K.

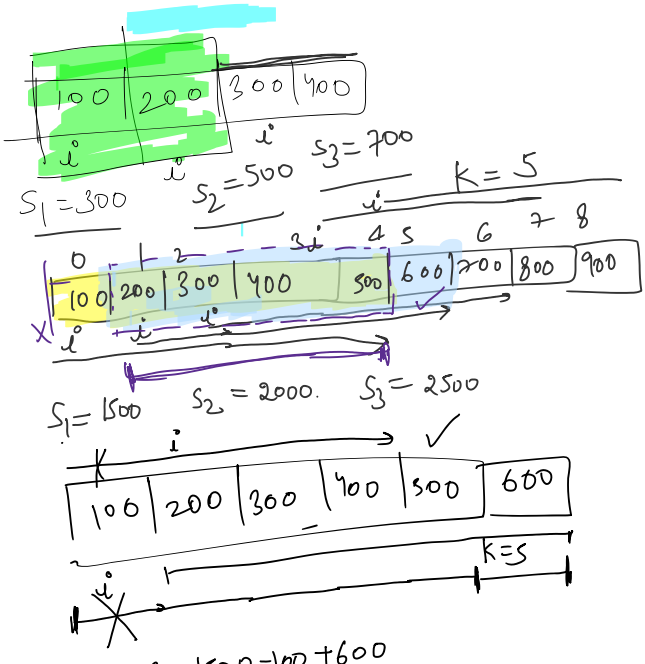
Example 1:

Input:
N = 4, K = 2
Arr = [100, 200, 300, 400]
Output:
700
Explanation:
$Arr_3 + Arr_4 = 700$,
which is maximum.

Array (N) | 100 | 200 | 300 | 400 |

K = 2

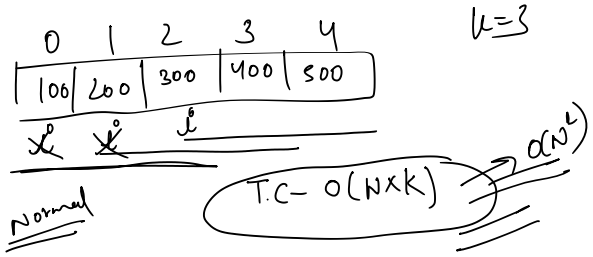
maximum sum

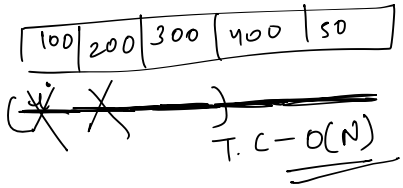


Array | String

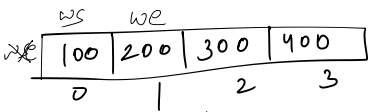
SubArray | Substring

Array / String
SubArray / SubString
K / window size
99 %





k=2 N=4



ws → window start
we → window end

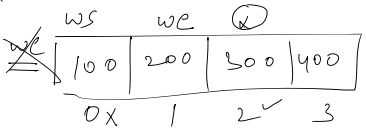
* we-ws+1 → window_length

we=0
ws=0
window_length = 0 -0+1 = 1
= 1 - 0 + 1 = 2

wsize = we - ws + 1 = 1
wsize = we - ws + 1 = 2

```
// ws = 0
// we = 0
int max = INT_MIN;
int sum = 0
while (____) {
    sum += A[we]; //
    if (we - ws + 1 < k) {
        we++;
    } else if (we - ws + 1 == k) {
        // compute output
        // window slide
    }
}
```

sum=0



sum = 100 + 200
= 300

k = 2

0 - 0 + 1
1 - 0 + 1 = 2

// output compute
maxi = max(maxi, sum);

// slide the window
sum -= A[ws];
ws++;
we++;

ws ws ws ws we k = 2

100	200	300	400
0	1	2	3

sum = 300 + 400 = 700

while (we < N){

sum += A[we]

// 0 - 0 + 1
// 1 - 0 + 1 = 2

if (we - ws + 1 < k){
we++;
}
else if (we - ws + 1 == k){
maxi = max(maxi, sum);
sum -= A[ws]
ws++;
we++;
}

maxi = 300 → 700

k = 3

100	200	300	400	500	600	100
0	1	2	3	4	5	6

we

// Initial ws = 0, we = 0
sum = 0

0 - 0 + 1 = 1
1 - 0 + 1 = 2

while (we < N){
sum += A[we]

sum = 100 + 200 + 300 - 100 + 400 + 500
sum = 930 - 30 = 900

maxi = 300 → 600 → 930 → 1500

1200
- 500
700

+600
1500
-400
1100

```
int we = 0, ws = 0;
while(we < N){
    // cumulative computation
    if(we- ws + 1 < K){
        // window size of aage badjhana h
    }
    else if(we-ws+1 == K){
        // output compute
        // window slide
    }
}
```

```cpp
long maximumSumSubarray(int K, vector<int> &Arr , int N){
    int we = 0, ws = 0;
    long maxi = INT_MIN;
    long sum = 0;
    while(we < N){
        // cumulative computation
        sum += Arr[we];
        if(we- ws + 1 < K){
            // window size of aage badjhana h
            we++;
        }
        else if(we-ws+1 == K){
            // output compute
            maxi = max(maxi,sum);
            // window slide
            sum -= Arr[ws];
            ws++;
            we++;
        }
    }
    return maxi;
}
```

T.C – O(N)
S.C – O(1)

1876. Substrings of Size Three with Distinct Characters

Hint

Easy 924 27

Companies

A string is **good** if there are no repeated characters.

Given a string `s`, return *the number of* ***good substrings*** *of length* ***three*** *in* `s`.

Note that if there are multiple occurrences of the same substring, every occurrence should be counted.

A **substring** is a contiguous sequence of characters in a string.

Example 1:

```
Input: s = "xyzzaz"
Output: 1
Explanation: There are 4 substrings of size 3: "xyz", "yzz", "zza", and "zaz".
The only good substring of length 3 is "xyz".
```

sliding window

String
substring
K=3

x y z z a z

a a b a b c a b c

K=3

	a	a	b	a	b	c	a	b	c
	0	1	2	3	4	5	6	7	8

0-0+1=1 1-0+1=2 2-0+1=3

char : count

b : 1
c : 1

map

count = 4

जब भी count 0 होगा तो map से delete कर देंगे character

```cpp
int countGoodSubstrings(string s) {
    int ws = 0, we = 0, n= s.size();
```

```cpp
int countGoodSubstrings(string s) {
    int ws = 0, we = 0, n= s.size();
    unordered_map<char,int> counter;
    int count = 0;
    while(we < n){
        counter[s[we]]++;
        if(we - ws + 1 < 3){
            we++;
        }
        else if(we - ws + 1 == 3){
            // answer find
            if(counter.size() == 3){
                count++;
            }
            // window slide
            // remove window start wala part
            counter[s[ws]]--;
            if(counter[s[ws]] == 0){
                counter.erase(s[ws]);
            }
            ws++;
            we++;
        }
    }
    return count;
}
```

T.C - O(N)
S.C - O(K)

2379. Minimum Recolors to Get K Consecutive Black Blocks

Hint

Easy 384 12

Companies

You are given a **0-indexed** string `blocks` of length `n`, where `blocks[i]` is either `'W'` or `'B'`, representing the color of the `i`th block. The characters `'W'` and `'B'` denote the colors white and black, respectively.

You are also given an integer `k`, which is the desired number of **consecutive** black blocks.

In one operation, you can **recolor** a white block such that it becomes a black block.

Return *the **minimum** number of operations needed such that there is at least **one** occurrence of `k` consecutive black blocks.*

Example 1:

```
Input: blocks = "WBBWWBBWBW", k = 7
Output: 3
Explanation:
One way to achieve 7 consecutive black blocks is to recolor the 0th, 3rd, and 4th blocks
so that blocks = "BBBBBBBWBW".
It can be shown that there is no way to achieve 7 consecutive black blocks in less than 3
operations.
Therefore, we return 3.
```

T.C - O(N)
S.C - O(1)

blocks = "WBBWWBBWBW" k = 7

wc = 3

WBW BBBW

wc = 0 wc = 1 k = 2

438. Find All Anagrams in a String

Medium 9.3K 287

Companies

Given two strings `s` and `p`, return *an array of all the start indices of `p`'s anagrams in `s`*. You may return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: s = "cbaebabacd", p = "abc"
Output: [0,6]
Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
The substring with start index = 6 is "bac", which is an anagram of "abc".
```

Example 2:

```
Input: s = "abab", p = "ab"
Output: [0,1,2]
Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
```

Anagram

fat ⟺ atf ⟺ taf

p ≠ anagram s ≠ ~~1 2 3~~ starting index

c d a e b a b a c d abc
0 1 2 3 4 5 6 7 8 9
[6]

a b a b
0 1 2 3

ab
[0, 1, 2]
answer

"abc"
window-size = p.size()
count = 3

pattern
a : 1
b : 1
c : 1

ws we
c b a e b a b a c d
0 1 2 3 4 5 6 7 8 9

cumulative map
c : 1
b : 1
a : 1

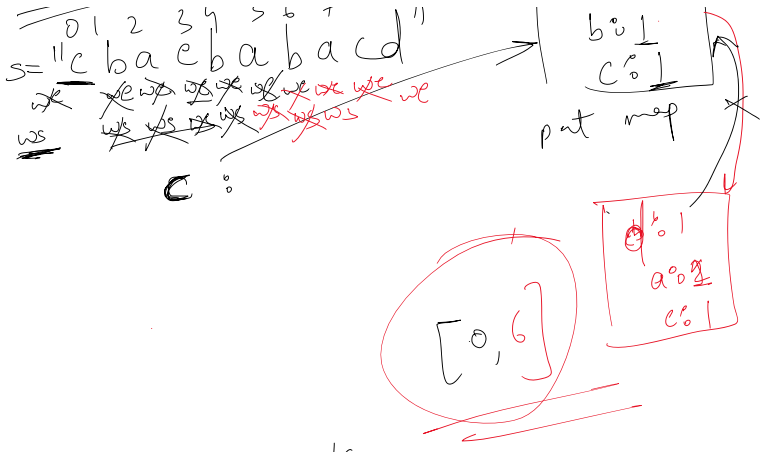
(4) p = aab
count = ~~2~~ 2 1 0
a : ~~1~~ 0 -1
b : 1 0
c : 0

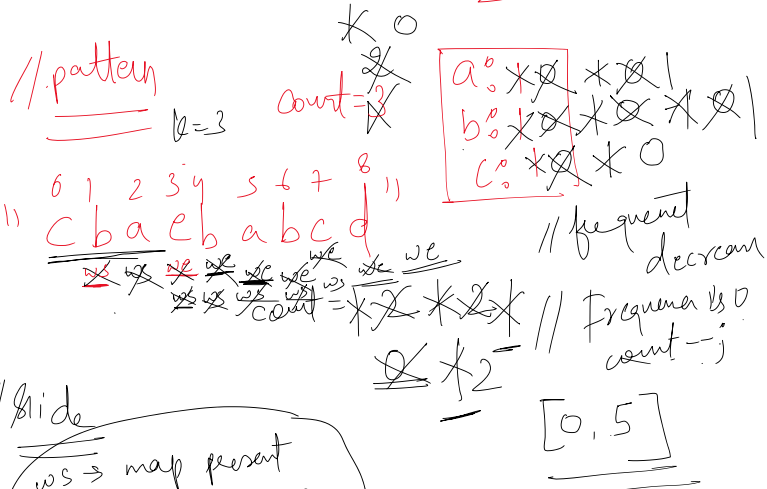
ws ~~we~~ we
a b c a z

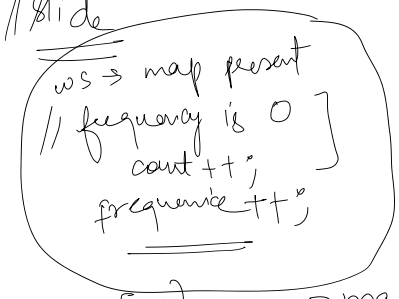
[0]

hashmap panel k = 3
0 1 2 3 4 5 6 7 8 9
s = "c b a e b a b a c d"
we

a : 1
b : 1
c : 1







s[we] → map

frequency--;

if frequency == 0

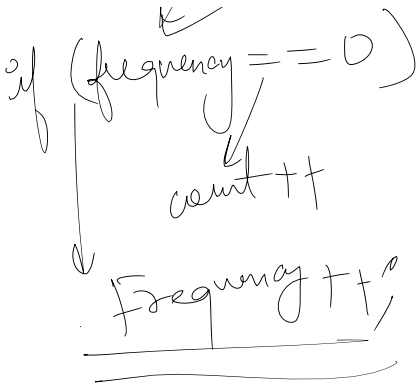
count--;

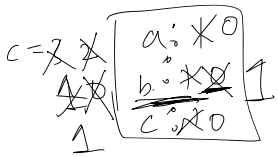
if (count == 0

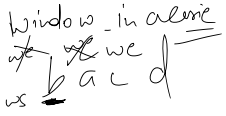
ws → array

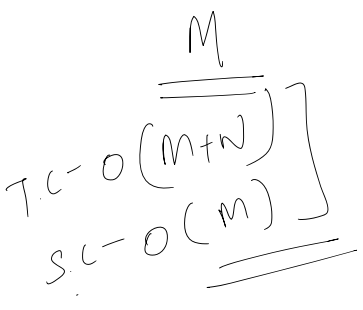
ws → map

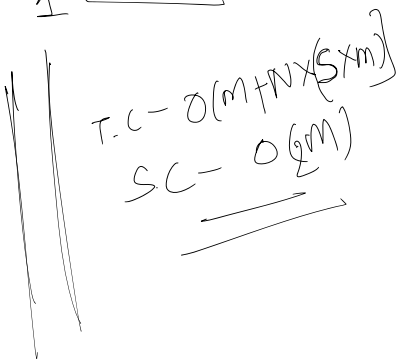
if (frequency == 0)

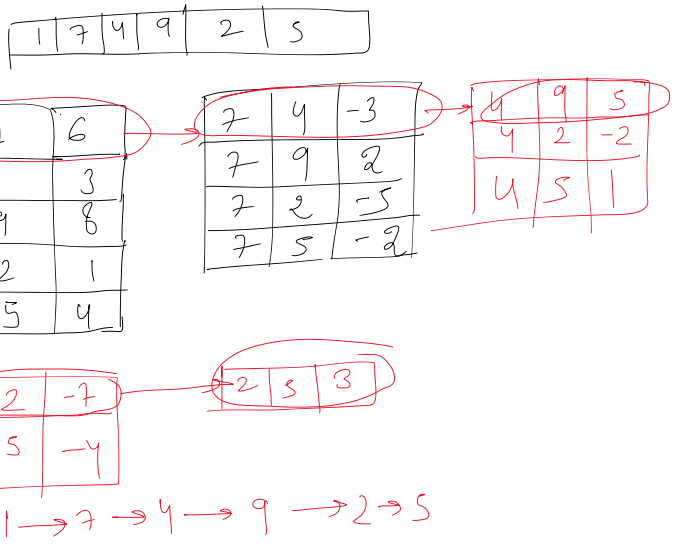

if (frequency == 0)
count ++
Frequency ++;

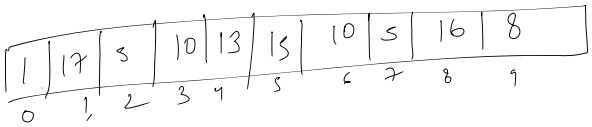

a: *0
b: **
c: *0

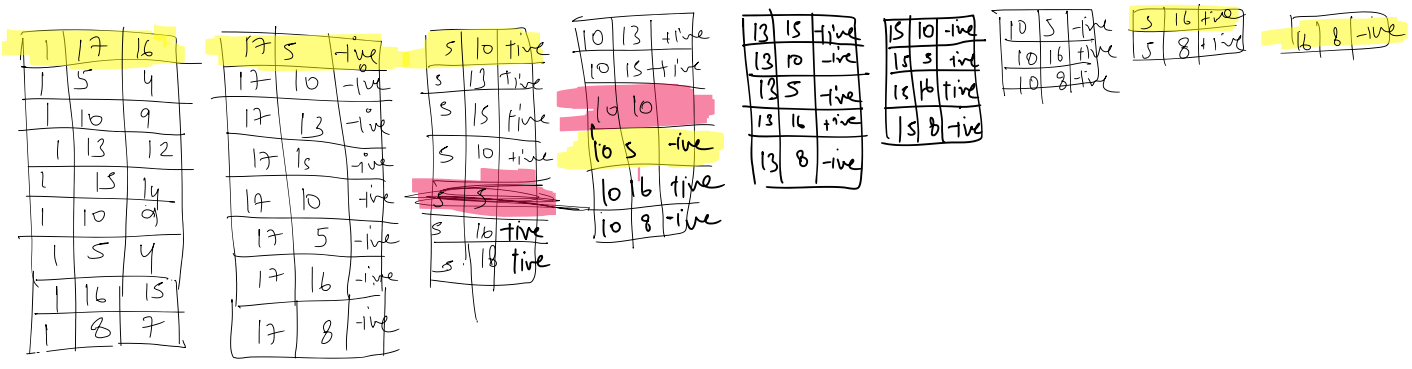

window_in_arene
we
b a c d


M
T.C- O(M+N)
S.C- O(M)


T.C- O(M+N*(S*m))
SC- O(qM)


1 7 4 9 2 5
1 7 6
1 4 3
1 9 8
1 2 1
1 5 4
7 4 -3
7 9 2
7 2 -5
7 5 -2
4 9 5
4 2 -2
4 5 1
9 2 -7
9 5 -4
2 5 3
1 → 7 → 4 → 9 → 2 → 5


1 17 5 10 13 15 10 5 16 8
0 1 2 3 4 5 6 7 8 9


1 17 16
1 5 4
1 10 9
1 13 12
1 15 14
1 10 9
1 5 4
1 16 15
1 8 7
17 5 -ive
17 10 -ive
17 13 -ive
17 15 -ive
17 10 -ive
17 5 -ive
17 16 -ive
17 8 -ive
5 10 +ive
5 13 +ive
5 15 +ive
5 10 +ive
5 5
5 16 +ive
5 18 +ive
10 13 +ive
10 15 +ive
10 10
10 5 -ive
10 16 +ive
10 8 -ive
13 15 +ive
13 10 -ive
13 5 -ive
13 16 +ive
13 8 -ive
15 10 -ive
15 5 -ive
15 16 +ive
15 8 -ive
10 5 -ive
10 16 +ive
10 8 -ive
5 16 +ive
5 8 +ive
16 8 -ive

1 → 17 → 5 → 10 → 5 → 16 → 8

1	17	5	10	13	15	10	5	16	8

bottom to top

s	D	
1	17	+
1	5	+
1	10	+
1	13	+
1	15	+
1	10	+
1	5	+
1	16	+
1	8	+

17	5	−
17	10	−
17	13	−
17	15	−
17	10	−
17	5	−
17	16	−
17	8	−

5	10	+
5	13	+
5	15	+
5	10	+
5	5	X
5	16	+
5	8	+

10	13	+
10	15	+
10	10	X
10	5	−
10	16	+
10	8	−

13	15	+
13	10	−
13	5	−
13	16	+
13	8	−

15	10	−
15	5	−
15	16	+
15	8	−

10	5	−
10	16	+
10	8	−

5	16	+
5	8	+

16	8	−

1 → 17 → 5 → 10 → 5 → 16 → 8

1	17	5	10	13	15	10	5	16	8

top to bottom

8	16	+
8	5	−
8	16	+
8	15	+
8	13	+
8	10	+
8	5	−
8	17	+
8	1	−

16	5	−
16	10	−
16	15	−
16	13	−
16	10	−
16	5	−
16	17	+
16	1	−

5	10	+
5	15	+
5	13	+
5	10	+
5	5	X
5	17	+
5	1	−

10	15	+
10	13	+
10	10	X
10	5	−
10	17	+
10	1	−

15	13	−
15	10	−
15	5	−
15	17	+
15	1	−

13	10	−
13	5	−
13	17	+
13	1	−

10	5	−
10	17	+
10	1	−

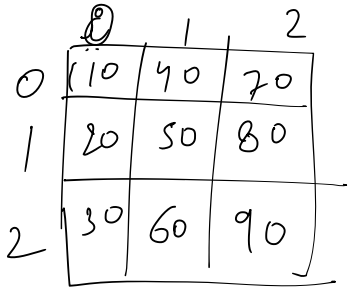
5	17	+
5	1	−

17	1	−

8 → 16 → 5 → 10 → 5 → 17 → 1

8 → 5 → 10 → 5 → 17 → 1

i


0 1 2
0 10 40 70
1 20 50 80
2 30 60 90