Lec-45

27/10/24 10:04 AM

Sliding window → $\frac{\lambda}{\times}$

Prob to identify

(iii) → fixed size window

→ variable size window

Array / String

input array / string → R

output ⇒ subarray / substring

condition

(i) fixed size | (ii) varible size

K-size subarray & sum maximum

[2, 3, 4, 5, 1, 7, 6, 4, 2, 3] k = 3

i=5

1-2=-1
5-3=2
4-2=1

n×k

(i) 1st window ans

0 1 2 3 4 5 6 7 8 9

Sum = 9+5−2 = 12+3 = 10+7−4 = 13+6−5

= 14+4−1 = 17+2−7 = 12+3−6 = 9

(ii) → (iii) window slow, window shrink, ans calc

2 3 4
3 4 5 S17
4 5 1
1 7 6
7 6 4
6 4 2
4 2 3

ans = 8 13 17 19

```
public static int Maximum_Sum(int[] arr, int k) {
        int sum = 0, ans = 0;
        // 1st window ka answer
        for (int i = 0; i < k; i++) {
            sum += arr[i];// sum = sum + arr[i];
        }
        ans = sum;
        // 2nd steps
        for (int i = k; i < arr.length; i++) {
            // window ko grow krna
            sum += arr[i];
            // window ko shrink krna
            sum -= arr[i - k];
            // ans update krna
            ans = Math.max(ans, sum);
        }
        return ans;
    }
```

0 1 2 3 4 5 6 7 8 9

i−k
3−2=1
4−2=2
5−2=3

Sum = 9+6−2 = 13
= 13+1−3 = (11)
= 11+7−4
= (14)

Variable size window

{[1, 3, 3, 4, 2]} k = 10

n^2

c = 12−13+1 = 12

si, ei

p = 2

p = p×arr[ei]

while (p>=k) {
  p = p / arr[si];
  si++;
}

ans = ei−si+1

ei++

```
public static int Product_Less_Than_K(int[] arr, int k) {
        int count = 0, si = 0, ei = 0, p = 1;
        while (ei < arr.length) {
            // grow
            p = p * arr[ei];
            // shrink
            while (p >= k) {
                p = p / arr[si];
                si++;
            }
            // ans update
            count += (ei - si + 1);
            ei++;
        }
        return count;
    }
```

si si si Sr

1 1 2 3 k=0

ei

p = 1

p = 1/1

p = 1/2 = 0

p = 0/3 = 0

aaabaab aab k=2

a

b max len

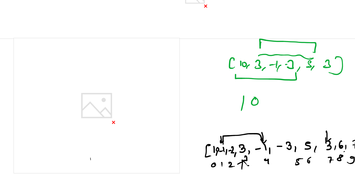
a min

int fil p = 1

aaabaab aab k=2

ans = 2 2 3 4 5 6 7 8

while (ei < size-len) {
  // 1. grow
  if (s.charAt(ei) == ch) {
    fil p++
  }
  // 2. shrink
  while (fil p > k && si < ei) {
    if (s.charAt(si) == ch) {
      fil p--
    }
    si++
  }
  // ans update
  ans = max(ans, ei-si+1)
  ei++
}

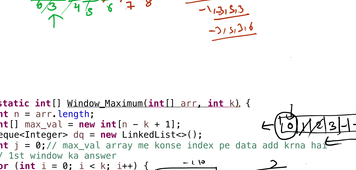


[4, -1, 2, 3, -1, 5, 3]

10

[4, -1, 2, 3, -1, -3, 5, 3, 6, 7] k = 4

0 1 2 3 4 5 6 7 8 9



[4, -1, 2, 3, -1, -3, 5, 3, 6, 7]

0 1 2 3 4 5 6 7 8 9

[4, -1, 2, 3, -1]

-2, 3, -1, -3

3, -1, -3, 5

-1, -3, 5, 3

-3, 5, 3, 6

i = 4−1 = 0 10

5−2 = 3 3

6−2 = 4 5

7−4 = 3 5

8−4 = 7 6

7

```
public static int[] Window_Maximum(int[] arr, int k) {
        int n = arr.length;
        int[] max_val = new int[n - k + 1];
        Deque<Integer> dq = new LinkedList<>();
        int j = 0;// max_val array me konse index pe data add krna ha
        // 1st window ka answer
        for (int i = 0; i < k; i++) {
            while (!dq.isEmpty() && arr[dq.getLast()] <= arr[i]) {
                dq.removeLast();
            }
            dq.add(i);
        }
    }
```

[10, 1, 2, 3, -1, 5]

[4, -1, 2, 3, -1, -3, 5, 3, 6, 7] k = 4

0 1 2 3 4 5 6 7 8 9