August 10, 2025

# TWO POINTERS FORM 2 & 3

## Homework Prblm 1.

Find count of # of subarrays with # of distinct element ==K.

Hint:

F(Arr, K-1) : # of subarrays with # of distinct element <=K-1

F(Arr, K) : # of subarrays with # of distinct element <K

The answer can be represented using the relation:

Cont of # subarr w # of dist.ele==K : F(Arr, K) − F(Arr, K-1)

→ Variation: Find max length of subarray with # of dist. element ==K

Hint: Similar to ques 2 of TP Form 1, only difference is before taking the max, it needs to be checked if freq == distinct.

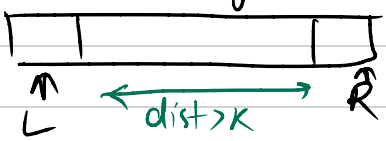
```
while(tail<n){
    while(head+1<n && check(arr[head+1])<=k)
    {
        head++;
        // insert head.
        insert(arr[head]);
    }
    // update answer
    if(distinct==k)
        ans = max(ans, head-tail+1);

    // remove element from tail
    if(tail<=head){
        // remove from DS.
        erase(arr[tail]);
        tail++;
    }else{
        tail++;
        head = tail-1;
```

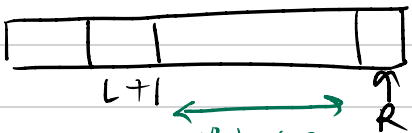
## Homework Prblm 2.

Find sum of length of all subarray with # of distinct element >K.

- # of distinct element >K, If this condition stands for (Part 1)

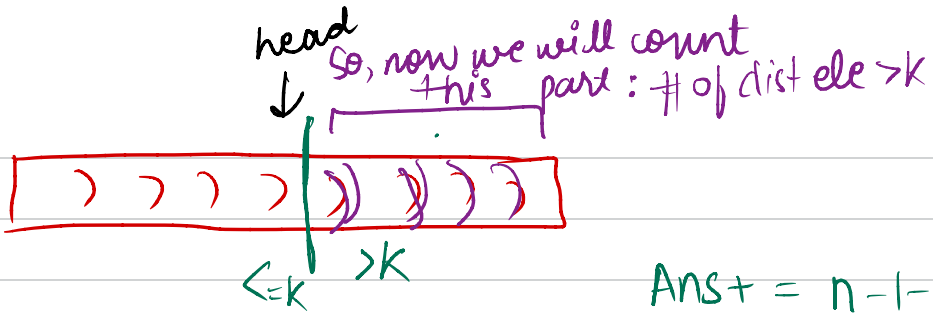



Then will it hold for L+1 to R




dist>K? Not necessarily True

AlgoZenith




Ans+ = n-1-head

(n-1-head) will give us subarray that ends after the above partioning. This is what we can add to the answer resulting in count of no of subarrays with dist ele > K.

→ Count how many are <= k using Two pointers and substract that from Total

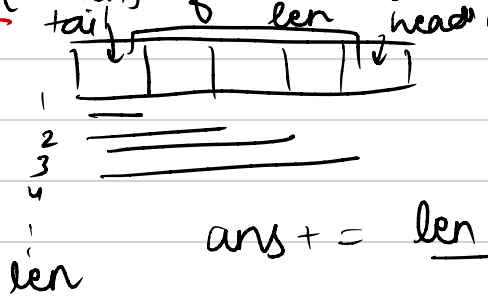
```
// update answer
ans += (n-start)-(tail-start+1);
```
>k  Total  <=k

```
int ans = 0;
int head=-1,tail=0;
while(tail<n){
    while(head+1<n && check(arr[head+1])<=k)
    {
        head++;
        // insert head.
        insert(arr[head]);
    }
    // update answer
    ans += ((n-1)-tail);

    // remove element from tail
    if(tail<=head){
        // remove from DS.
        erase(arr[tail]);
        tail++;
    }else{
```

(Part 2) Find the sum of lengths of these subarrays.

Count( $\Sigma$ len) of all subarrays with dist<=K




len = tail - head + 1

$$ans += \frac{len\,(len+1)}{2}$$

```
long long ans = 0;
int head=-1,tail=0;
while(tail<n){
    while(head+1<n && check(arr[head+1])<=k)
    {
        head++;
        // insert head.
        insert(arr[head]);
    }
    // update answer
    long long temp = tail-head+1;
    ans += temp*(temp+1)/2;

    // remove element from tail
    if(tail<=head){
        // remove from DS.
        erase(arr[tail]);
        tail++;
    }else{
```

AlgoZenith

# * Form 2 : Two pointers moving in opposite direction [2Sum, 3Sum]

**Ques 1.** Find max value of $[\min(a[i], a[j]) * (j-i)]$ for $0 \le i < j \le N$

| 3 | 5 | 8 | 3 | 2 | 7 | 1 |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

$N \le 10^5$
$A[i] \le 10^9$

Ex:
- $i=1, j=5$
  $\min(5,7) = 5$, $ans = 5*4 = 20$
- $i=2, j=5$
  $\min(7,8) = 7$, $ans = 7*3 = 21$

Similar Problem : https://leetcode.com/problems/container-with-most-water/description/

$[\min(a[i], a[j]) * (j-i)]$
(min, length (gap))

Approach for our solution ⇒

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 3 | 5 | 8 | 3 | 2 | 7 | 1 |

## Code solution of mentioned leetcode problem:

```
int maxArea(vector<int>& height) {
    int i=0,j=height.size()-1;
    int ans = 0;
    while(i<j){
        ans = max(ans, min(height[i],height[j])*(j-i));
        if(height[i]<height[j]){
            j--;
        }else{
            i++;
        }
    }
```

For $i=0, j=6$

- length = 6
  and for length = 6, there is only 1 option → [0,6]
  $\min(a[0], a[6]) = 1$
  $ans = min * len = 6$ → Better choice to maximise ans
- length = 5 [0,5] [1,6]
  $\min(a[0], a[5]) = 3$
  $ans = min * len = 3*5 = 15$

decreasing length

If length is decreasing then min has to increase to give a better answer.

| 3 | 5 | 8 | 3 | 2 | 7 | 1 |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

L R

Here, we are seeing 2 pointers at 3 and 1. Now if we have to move a pointer then we will move R since 1 is smaller & we are trying to increase our min.

Now, let's say we want to see for length = 4, then current L & R, which will move? L will move since 3 is smaller.

- length = 4 [1, 5] as shown above
  $\min(a[1], a[5]) = 5$
  $ans = 5*4 = 20$

Continue

Since 5 is smaller than 7, therefore, we will move the L pointer for length = 3

- length = 3 [2,5]
  $\min(a[2], a[5]) = 7$, Ans = 3*7 = 21
- length = 2 [2,4]
  $\min(a[2], a[4]) = 2$
  Ans = 2*2 = 4

Here, this is probably not the most optimal answer but we have to increase our min if we decrease our length.

* FORM 3

Problem link : https://leetcode.com/problems/is-subsequence/description/

Ques 1. Is S a subsequence of T

Example:

T = a c d b a c d a b  Size = N

S = c b a d b  Size = M

Ans = Yes, S is a subsequence of T

Approach for the solution : O(N+M)

We will have 2 pointers for both sequence

T = a c d b a c d a b
    ↑P1

S = c b a d b
    ↑P2

c ≠ a

1st iteration : First, we are looking for c in T

But, $P_1$ points to a. This a can never help us get into any sequence. So we can simply discard it.

T = ~~a~~ c d b a c d a b
    ↑P1

S = c b a d b
    ↑P2

2nd iteration : c == c, Move both $P_1$, $P_2$

T = ~~a~~ c d b a c d a b
    ↑P1

S = c b a d b
    ↑P2

Now, the problem is reduced to find the subsequence in green from S in green box of T.

3rd iteration : b ≠ d, Move $P_1$ and discard d.

T = ~~a~~ c ~~d~~ b a c d a b
    ↑P1

S = c b a d b
    ↑P2

4th iteration : b = b, Move $P_1$ & $P_2$

T = ~~a~~ c ~~d~~ b a c d a b
    ↑P1

S = c b a d b
    ↑P2

5th iteration : a = a, Move $P_1$ & $P_2$

T = ~~a~~ c ~~d~~ b a c d a b
    ↑P1

S = c b a d b
    ↑P2

6th iteration : c ≠ d, Move $P_1$

T = ~~a~~ c ~~d~~ b a ~~c~~ d a b
    ↑P1

S = c b a d b
    ↑P2

7th iteration : d = d, Move $P_1$ & $P_2$

T = ~~a~~ c ~~d~~ b a ~~c~~ d a b
    ↑P1

S = c b a d b
    ↑P2

8th iteration : a ≠ b, Move $P_1$

T = ~~a~~ c ~~d~~ b a ~~c~~ d ~~a~~ b
    ↑P1

S = c b a d b
    ↑P2

Final iteration : b = b, Move $P_1$ & $P_2$

But $P_1$ & $P_2$ reaches end.

Hence program terminates.

Output : Yes, S is a subsequence of T.

If $P_2$ is not able to traverse the whole string then we don't have a match

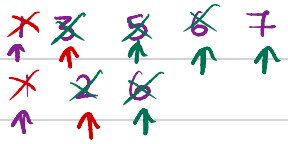
Code for the leetcode problem :

```
bool isSubsequence(string s, string t) {
    int i = 0, j = 0;
    while(i<s.length() && j<t.length()){
        if(s[i]==t[j]){
            i++;
            j++;
        }else{
            j++;
        }
    }
    if(i==s.length())return true;
    else return false;
}
```

Example that follows the above discussed approach:

- Merge Sort.

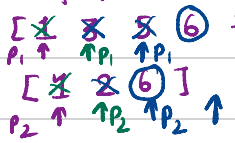
Have 2 pointers. whichever is small, append to answer list, discard that ele and move the pointer.



Ans: [1,1,2,3,5,6,6,7]

- Intersection of 2 sorted list (Union is also possible)

2 pointers, one for each set, if they are same then append that element in ans list & move both pointers forward. If they are not same, discard the smaller one & move the assigned pointer forward. If one of the list gets over then ans list is the output.



Ans: [1,6]