Todays Content

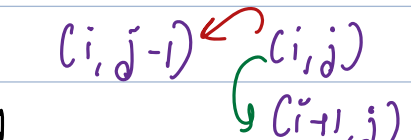
1. Search k in a row-wise sorted matrix
2. length of longest subarray with distinct elements $\leq k$.

Q1 Given rowwise columnwise sorted matrix[N][N] find k?

Constraints:

$1 <= N, M <= 10^3$

$-10^6 <= mat[i][j] <= 10^6$



Eg: mat[6][6] k = 12, 18, 21

↓	0	1	2	3	4	5
0	-10	-5	-2	2	4	7<12
1	-7	-1	3	4	6	9<12
2	-2	3	5	7	11<12	14>12
3	3	6	8	11<12	14>12	17
4	7	11	12	15>12	19	20
5	10<12	14	18	20	24	29

#Ideas1 Iterate in mat[][] & search

TC: O(N*M) SC: O(1)

#Ideas2 a. Apply BS for each row

TC: O(N * log M)

#BS = Binary Search

b. Apply BS for each col

TC: O(M log N)

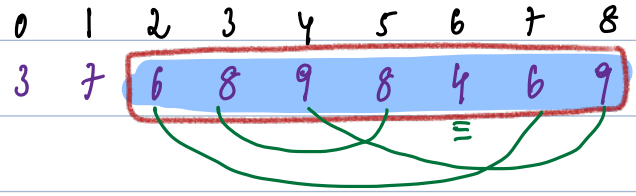
#Ideas3: Apply 2 pointer from Top-Right or Bottom left

```
boolean matSearch(int mat[][], int N, int M, int k){
    int i=0, j=M-1;
    while(i<N && j>=0){
        if(mat[i][j]==k){
            return True;
        }
        else if(mat[i][j]<k){
            #Discard row
            i++;
        }
        else{ #mat[i][j]>k Discard column
            j--;
        }
    }
    return false;
}
```

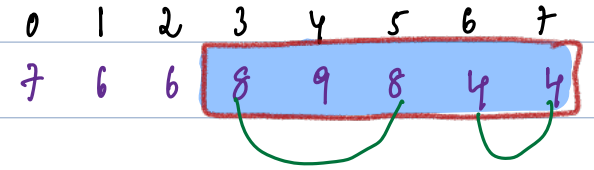
#Note: In each iteration we skip a row or column, total it will have N+M iterations

Q2: Given an ar[N] elements, find out length of longest subarray with number of distinct elements <= k

Ex1: ar[10] = { 3 7 6 8 9 8 4 6 9 12 } len = 7.
k = 4



Ex2: ar[10] = { 7 6 6 8 9 8 4 4 7 3 } len = 5
k = 3



#Ideal:

Generate all subarrays (#subarrays) TC: O(N^2 * N) -> Insert subarray in set SC: O(N)

For each subarray calculate no: of distinct element?

Insert subarray in hashset/ hashmap & get set size

if(size <= k) { ans = max(ans, #subarray length) }

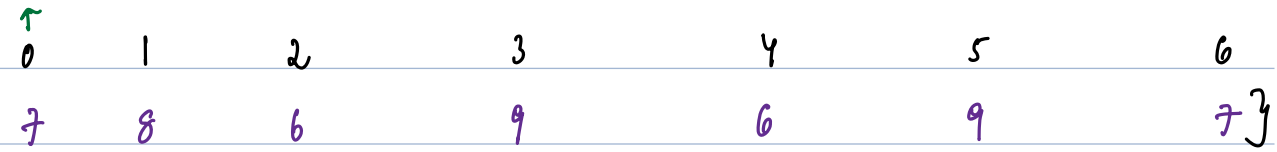
return ans;

#Idea2: Generate all subarrays TC: O(N^2 * 1) = O(N^2) SC: O(N)

For each subarray calculate no: of distinct element?

Using previous subarray information? How?

#By carry forwarding hashset



hs : { } {7} {7 8} {7 8 6} {7 8 6 9} {7 8 6 9} {7 8 6 9} {7 8 6 9}

#start: 0 1 2 3 4 4 4 4

hs : { } {8} {8 6} {8 6 9} {8 6 9} {8 6 9} {8 6 9 7}

#start: 1 1 2 3 3 3 4

Ideas:

Target: length of longest subarray len with distinct ele $x = k$.

Search space: l h

{0 N]

M > 6

	0	1	2	3	4	5	6	7	8	9
ar[10] = {	7	6	6	8	9	8	4	4	7	3]

k = 4

l h m

0 10 5; #Check if there exists a subarray of len = 5; with no: of distinct ele $x = 4$. # True

5 # of subarray of len = 5 with

T T T [T] distinct $x = 4$ exists, it's a

ans = 5; guarantee that subarray of

l = m+1; len < 5, with distinct $x = 4$ exists,

6 10 8; #Check if there exists a subarray of len = 8; with no: of distinct ele $x = 4$.

8

[F] F F F ..

h = m-1;

#Search Space: T T T T .. T [T] F F F .. F F

Search Space = h - l + 1 = N + 1

```
int longest(vector<int>& arr, int k){
    int l=0, h=arr.size(), ans=0;
    while(l<=h){
        int m=(l+h)/2;
        if( check, if there exists a subarray len=m, with distct<=k)
            ans=m;
            l=m+1;
        else{
            h=m-1;
        }
    }
    return ans;
}
```

TC: $O(N \log_2 N+1)$

Approach: Using Sliding window

TODO

TC: O(N)

ideay:

k = 3

	0	1	2	3	4	5	6	7	8	9	10
ar[10] = {	~~k~~	~~k~~	~~k~~	~~k~~	~~k~~	~~k~~	4	4	7	3	}
							P_1				P_2

#Note:
Counting distinct with hashset will not work need hashmap

Valid ans hm: { 4:2 3:1 7:1 }

6:1 7:1 9:1 8:0

P_1	P_2	Dist <= k		
0	-1	0 <= k	0	P_{2++} In ar[P_2] in hm
0	0	1 <= k	1	P_{2++} In ar[P_2] in hm
0	1	2 <= k	2	P_{2++} In ar[P_2] in hm
0	2	2 <= k	3	P_{2++} In ar[P_2] in hm
0	3	3 <= k	4	P_{2++} In ar[P_2] in hm
0	4	4 <= k	*	Del ar[P_1] in hm P_{1++}
1	4	4 <= k	*	Del ar[P_1] in hm P_{1++}
2	4	3 <= k	4	P_{2++} In ar[P_2] in hm
2	5	3 <= k	4	P_{2++} In ar[P_2] in hm
2	6	4 <= k	*	Del ar[P_1] in hm P_{1++}
3	6	3 <= k	4	P_{2++} In ar[P_2] in hm
3	7	3 <= k	5	P_{2++} In ar[P_2] in hm
3	8	4 <= k	*	Del ar[P_1] in hm P_{1++}
4	8	4 <= k	*	Del ar[P_1] in hm P_{1++}
5	8	3 <= k	5	P_{2++} In ar[P_2] in hm
5	9	4 <= k	*	Del ar[P_1] in hm P_{1++}
6	9	3 <= k	5	P_{2++}; P_2 outside stop & retu ans = 5.

TC: O(N+N) = O(N) SC: O(N)

```
int longest(vector<int> &ar, int k){
    int N = ar.size();
    int P1=0, P2=-1; # Sub [P1 P2]; #Empty subarray No Initialization
    unordered_map hm;
    int ans=0, N=ar.size();
    while(P2<N){
        if( hm.size()<=k){ # [P1 .. P2] subarray is valid.
            ans = max(ans, P2-P1+1); # update ans
            P2++;                    # updating P2
            if(P2==N){break}
        } hm[ar[P2]]++;              # Adding new ar[P2] is subarray
        else{ # [P1 - P2] is invalid, [P1.. P2, P2+1 P2+2.. N-1]

            hm[ar[P1]]--; # Deleting ar[P1] from subarray
            if(hm[ar[P1]]==0){hm.erase(ar[P1])}
            P1++; # update P1
        }
    }
}
```

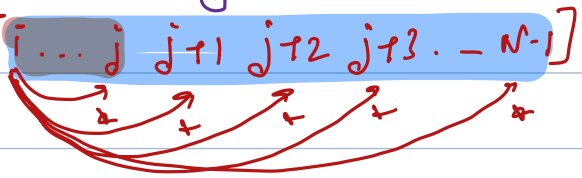
#2 Pointers on Subarrays:

Type 1: 2 Pointers:

If let's say [i..j] is invalid subarray:

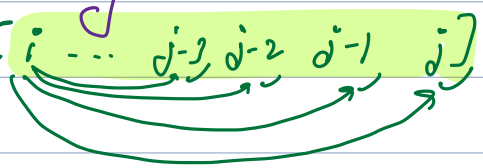
If we can show that [i ... j j+1 j+2 j+3 ... N-1]

Other way to write above



If let's say [i..j] is valid subarray

If we can show that [i ... j-3 j-2 j-1 j]



Pseudo Code:

```
int p1=0, p2=-1;
int ans=0, N=ar.size();
while(p2<N){
    if( valid [p1..p2] ){
        update ans;
        p2++; ——→ # get next sub [i..j+1]
        if(p2==N){break}
    } Add ar[p2] in subarray → # New ele in subarray.
    else{
        # [i..j] invalid
        remove ar[p1] in subarray
        p1++;
    }
}
```