13/02/2026
Friday

Sliding window

Longest substring without repeating characters

s = cadbzabcd n = 9

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

Here characters are in Lowercase, uppercase and some are special characters

Approach - 1

Generate All substrings from the given substring and check whether all characters are distinct or not and update to max length

first for

c
ca
cad
cadb
cadbz
cadbza Here we get a 2 times
So we break here only bcuz no use in going further

similarly for

a
a d
a d b
a d b z
a d b z a break

then storing into max

Approach -2

Take 2 pointers

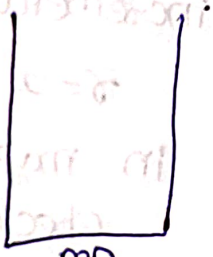
always try to take 2 pointers when u get substrings

	0	1	2	3	4	5	6	7	8
S =	c	a	d	b	z	a	b	c	d

Take a map and in that store characters and their indexes

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

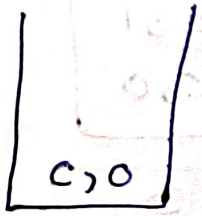
l
r



place l and r both at first index

Now check is c in map no

so we add



next cal len = r-l+1
= 0-0+1
= 1

maxlen = max(len, maxlen)
1, 0

maxlen = 1

Next increment r

r = 1

Now again check is s.charAt(r) is

in map

a is not in map

so add a

a, 1
c, 0

Cal len = r - l + 1

= 1 - 0 + 1

= 2

maxlen = max (len, maxlen)

2, 1

= 2

increment r

r = 2

in map

check is s.charAt(r) is these

d is not these

then add into map

d, 2
a, 1
c, 0

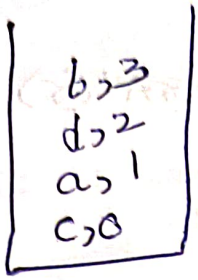
Cal len = 3

maxlen = 3

r++

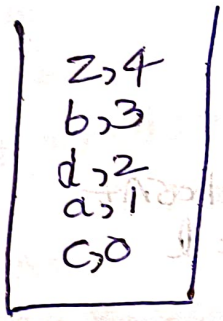
r = 3

Similarly
b is not present add



len=4
maxlen=4
r++
r=4

again
z is not present add



len=5
maxlen=5
r++
r=5

Now check is s.charAt(r) is present in map a is present in map

Now we will check where is l beuz if it is greater no need to change l

if that character position is greater than l

then we need to place l at that character position +1

l=0 → s.charAt(r)

mp.get(r) = 1

1 >= 0

so move l → s.charAt(r)

l = mp.get(r)+1

l = 2

maxlen = 5

update

z, 4
b, 3
d, 2
a, ~~1~~ 5
c, 0

increment r

r = 6

b is these

now check → s.charAt.

mp.get(r) >= l

3 >= 2

so move l

l = 4

maxlen = 5

z, 4
b, ~~3~~ 6
d, 2
a, ~~1~~ 5
c, 0

r++

r = 7

c is there

we check

mp.get(c) >= l

r >= 4

No

so no need to move

we just find len

len = r - l + 1

7 - 4 + 1

len = 4

maxlen = 5

r++

r = 8

z, 4
b, ~~3~~ 6
d, ~~2~~ 8
a, ~~0~~ 5
c, ~~0~~ 7

d is there

now check

mp.get(d) >= l

2 >= 4

No

len = 8 - 4 + 1

= 5

maxlen = 5

q < q fails

return max length

5

Code

```
public int solve (String s) {
    int n = s.length();
    Map<Character, Integer> mp =
                    new HashMap<>();

    int l=0, r=0; int len=0, maxlen=0;

    while (r<n) {
        if (mp.containsKey (s.charAt(r)))
        {
            if (mp.get(s.charAt(r))>=l)
            {
                l = mp.get(s.charAt(r))+1;
            }
        }
        len = r-l+1;
        maxlen = Math.max(len, maxlen);
        mp.put (s.charAt(r), r);
        r++;
    }
    return maxlen;
}
```

Max Consecutive Ones III

Given a array nums binary array and integer k return the max no of consecutive 1's in array you can flip atmost k 0's.

this is same as
return the length of longest subarray that contains atmost k zero's

Approach-1

nums = [1,1,1,0,0,0,1,1,1,1,0] k=2

I will generate all subarrays and take only k 0's subarray and among them I will find max len

```
for(int i=0; i<n; i++)
{
    zeros=0;
    for(int j=i; j<n; j++)
    {
        if(nums[j]==0) zeros++;
        if(zeros<=k)
            {zeros++}
            len = j-i+1;
            maxlen = max(len, maxlen)
        else
            break
    }
}
return max length
```

Approach-2

nums = [1,1,1,0,0,0,1,1,1,1,0] k=2

In this approach

take 2 pointers subarrays right

```
 0 1 2 3 4 5 6 7 8 9 10
[1,1,1,0,0,0,1,1,1,1,0]   k=2
 l=0
 r=0
```

zeros=0
len=1
maxlen=1

next increment r

r=1

```
[1 1 1 0 0 0 1 1 1 1 0]   k=2
 l r
```

zeros=0
len=2
maxlen=2

r++
r=2

```
[1 1 1 0 0 0 1 1 1 1 0]   k=2
 l   r
```

zeros=0
len=3
maxlen=3

r++
r=3

[1 1 1 0 0 0 1 1 1 1 0] K=2
 l r

zeros=1
len=4
maxlen=4

r++

r=4

[1 1 1 0 0 0 1 1 1 1 0] K=2
 l r

zeros=2
len=5
maxlen=5

r++;

r=5

[1 1 1 0 0 0 1 1 1 1 0] K=2
 l r

zeros=3
len

But we need two only ←

so we remove from left and remove count zeros untill it is equal to k

[1 1 1 0 0 0 1 1 1 1 0] K=2
 l r

zeros=2
len=2
maxlen=5

r++

r=6

```
  0 1 2 3 4 5 6 7 8 9 10
[ 1 1 1 0 0 0 1 1 1 1 0 ]   K=2
    l       r
```

zeros = 2

len = 3

maxlen = 5

r++

r = 7

```
  0 1 2 3 4 5 6 7 8 9 10
[ 1 1 1 0 0 0 1 1 1 1 0 ]   K=2
      l       r
```

zeros = 2

len = 4

maxlen = 5

r++

r = 8

```
  0 1 2 3 4 5 6 7 8 9 10
[ 1 1 1 0 0 0 1 1 1 1 0 ]   K=2
        l       r
```

zeros = 2

len = 5

maxlen = 5

r++

r = 9

```
  0 1 2 3 4 5 6 7 8 9 10
[ 1 1 1 0 0 0 1 1 1 1 0 ]   K=2
          l       r
```

zeros = 2

len = 6

maxlen = 6

r++

r = 10

```
  0 1 2 3 4 5 6 7 8 9 10
[ 1 1 1 0 0 0 1 1 1 1 0 ]   k=2
      l         r

                zeros=3
```

we reduce
until it is equal to k
remove count zeros
if nums[l]=0

```
  0 1 2 3 4 5 6 7 8 9 10
[ 1 1 1 0 0 0 1 1 1 1 0 ]
      l         r

                zeros=2
                len=6
                maxlen=6
          r++
        r=11
```

fails.

```
int l=0, r=0, len=0, maxlen=0; zeros=0;
while (r<n) {
    if(nums[r]==0)
            zeros++;
    while (zeros>k)
    {
        if (nums[l]==0) zeros--;
            l++;
    }
    len= r-l+1;
    maxlen= Math.max (len, maxlen);
    r++;
}
return maxlen
```

T.C:- O(2N)
S.C:- O(1)

Approach-3

```
        0 1 2 3 4 5 6 7 8 9 10
nums = [1 1 1 0 0 0 1 1 1 1 0]   k=2
       l
       r
```

zeros=0
len=1
maxlen=1

r++
r=1

```
        0 1 2 3 4 5 6 7 8 9 10
       [1 1 1 0 0 0 1 1 1 1 0]   k=2
       l r
```

zeros=0
len=2
maxlen=2

r++
r=2

```
       [1 1 1 0 0 0 1 1 1 1 0]   k=2
       l   r
```

zeros=0
len=3
maxlen=3

r++
r=3

```
       [1 1 1 0 0 0 1 1 1 1 0]   k=2
```

zeros=1
len=4
maxlen=4

r++
r=4

```
  0 1 2 3 4 5 6 7 8 9 10
[ 1 1 1 0 0 0 1 1 1 1 0 ]   k=2
  l       r
```

zeros = 2
len = 5
maxlen = 5

r++

r = 5

```
  0 1 2 3 4 5 6 7 8 9 10
[ 1 1 1 0 0 0 1 1 1 1 0 ]   k=2
  l           r
```

zeros = 3

So I will move l to rightside
if it is 0 then
I decrease zeros

```
[ 1 1 1 0 0 0 1 1 1 1 0 ]   k=2
  l           r
```

zeros = 3
len

if (zeros <= k)
then only I will
find len
and store in max

if (zeros > k)
if (nums[l] == 0)
zeros--;
l++

```
[ 1 1 1 0 0 0 1 1 1 1 0 ]   k=2
    l         r
      l       r
        l     r
          l   r
            l r   len = 6
```

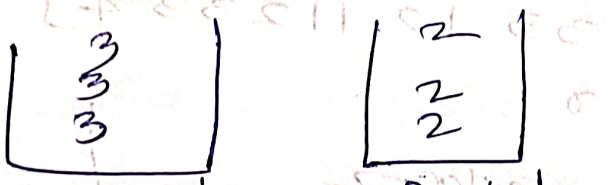
```
int n=nums.length
int l=0,r=0, len=0,maxlen=0,zeros=0
while(r<n)
{
    if(nums[r]==0) zeros++;
    if(zeros<=k){
        len= r-l+1
        maxlen= max(len,maxlen)
    }
    if(zeros>k){
        if(nums[l]==0)
            zeros--;
        l++
    }
    r++;
}
return maxlen
```

Fruits into Baskets

fruits = [1,2,1]

arr[] = [3 3 3 1 2 1 1 2 3 3 4]

Conditions are
you need to fill these into 2 baskets
each basket should contain only 1 item type items



Basket 1 Basket 2

Here you should not skip you should fill by taking in a row and fill it.

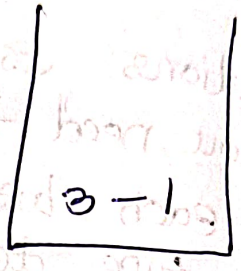
Approach-1

```
for(int i=0;i<n;i++){
    set<Integer> st
  for(int j=i;j<n;j++)
  {
        st.add(nums[j]);
        if(st.size<=2){
            len=r-l+1;
          maxlen= Max(len,maxlen)
        }
        else
        {
          break;
        }
    }
}
return maxlen
```

Approach-2

[3 3 3 1 2 1 1 2 3 3 4]
l
r

maxlen=1

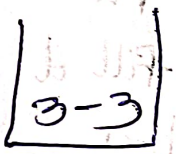


[3 3 3 1 2 1 1 2 3 3 4]
l r

maxlen=2

[3 3 3 1 2 1 1 2 3 3 4]
l r

maxlen=3

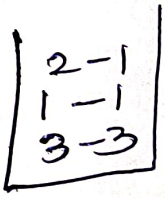


[3 3 3 1 2 1 1 2 3 3 4]
l r

maxlen=4

[3 3 3 1 2 1 1 2 3 3 4]
l r

Here size is 3
but we need
only 2
so we remove until
we
get size 2



[3 3 3 1 2 1 1 2 3 3 4]
 l r

len=2
maxlen=4

2-1
1-1
3-0

removed

[3 3 3 1 2 1 1 2 3 3 4]
 l r

len=3
maxlen=4

2-1
1-2

[3 3 3 1 2 1 1 2 3 3 4]
 l r

len=4
maxlen=4

2-1
1-3

[3 3 3 1 2 1 1 2 3 3 4]
 l r

len=5
maxlen=5

2-2
1-3

[3 3 3 1 2 1 1 2 3 3 4]
 l r

size>2
remove until
it is 2

3-1
2-2
1-3

[3 3 3 1 2 1 1 2 3 3 4]
 l r

len=2
maxlen=5

3-1
2-1

at last we get
len=4
maxlen=5

Approach-3

Instead of while we
use keeping if that only.

if (mp.size <= k/2)
then only storing

Approach-2 Code

```
Map<Integer, Integer> mp = new HashMap<>
int l=0, r=0, len=0, maxlen=0;
while(r<n){
    mp.put(nums[r], mp.getOrDefault
                    (nums[r],0)
                    +1);
    if(mp.size()<=2){
        len = r-l+1;
        maxlen = max(len, maxlen);
    }
while(mp.size()>2)
{
    if(mp.containsKey(nums[l]))
    {
        int f = mp.get(nums[l]);
        f--;
        mp.put(nums[l], f);
        if(f==0)
        { mp.remove(nums[l]);
        }
        l++;
    }
}
r++;
}
```

return maxLen;

Similarly for approach-3
just change while loop into
if block

Longest Repeating Characters Replacement

You are given a string s and an integer k.
You can choose any character of the
string and change it to any other uppercase
English character. You can perform this
operation at most k times

Approach-1
brute force
generate all substrings
e.g.-
S = AABABBA k=1

A

Here we can
make 1 conversion

for each substring right
we check the
length of the substring
and maxFreq
if their difference is less than
equal to k then it is possible
otherwise break it

```
Map<Character, Integer> mp = new HashM
for (int i=0; i<n; i++)
{
    maxFreq = 0
    for (int j=i; j<n; j++)
    {
        mp.put(s.charAt(j), 1 + mp.getOrDefau
                                (s.charAt(j), 0)
        maxFreq = Math.max(maxFreq,
                           mp.get
                            (s.charAt(j)
        if((j-i+1) - maxFreq <= k)
            maxLen = Math.max(maxLen, j-i+1);
        else
            break;
    }
}
```

Approach - 2

two pointers

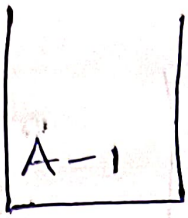
A A A B B C C D K=2

Same as brute force logic

but we use following 2-pointers

A A A B B C C D K=2

l r



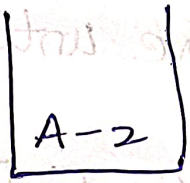
maxfreq=1 len=1

noofchanges=0

maxlen=1

A A A B B C C D

l r



maxfreq=2 len=2

noofchanges=0

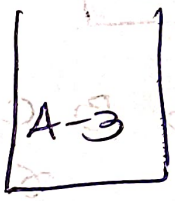
maxlen=2

A A A B B C C D K=2

l r

maxfreq=3

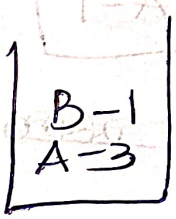
len=3



noofchanges=0

A A A B B C C D K=2

l r



maxfreq=3

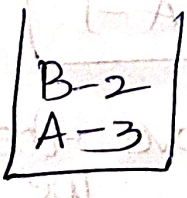
len=4

noofchanges=1

maxlen=4

A A A B B C C D

l r



len=5

maxfreq=3

noofchanges=2

maxlen=5

A A A B B C C D K=2
l r

C-1
B-2
A-3

maxfreq=3
len=6
noofchanges=3
3>2

So we remove untill we get K=2

A A A B B C C D
l r

C-1
B-2
A-2

maxfreq=2
len=5

A A A B B C C D
l r

len=4

C-1
B-2
A-1

maxfreq=2
noofchanges=2

~~So we remove~~ maxlen=5

A A A B B C C D
l r

C-2
B-2
A-1

len=5
maxfreq=2
noofchanges=3

remove then we get

A A A B B C C D
l r

C-2
B-2

len=4
maxfreq=2
roofcharges = 2
maxlen = 4

A A A B B C C D
l r

D-1
C-2
B-2

len=5
maxfreq=2
roofcharges=3

remove

A A A B B C C D
l r

D-1
C-2
B-1

len=4
maxfreq=2
roofcharges=2
maxlen=5

```
l=0, r=0, maxlen=0, maxfreq=0;
Map<characters, Integers> mp= new HashMap<>;
while (r<n){
    mp.put(s.charAt(r), 1+ mp.put(s.charAt(r)
                                  ,0));
    maxfreq= Math.max(maxfreq, mp.get
                      (s.charAt(r));
    roofcharges = (r-l+1)- maxfreq;
```

```
if(noofchanges <=k)
    maxlen= Math.max (maxlen, r-l+1);
while ((r-l+1)- maxfreq >k){
        if(mp.containskey (s.charAt(l))
        {
            int f= mp.get(f);
             f--;
            mp.put (s.charAt(l),f);
            if(f==0)
                mp.remove (s.charAt(l))
            for(char ele: mp.keyset())
            {
                if(mp.get(ele)>maxfreq)
                    maxfreq= mp.get(ele);
            }
        }
        l++;
    }
    r++;
}
return maxlen;
}
```