Longest substring with at most K distinct characters

s = a a a b b c c d k = 2

Longest substring with at most K distinct characters

s = a a a b b c c d k = 2

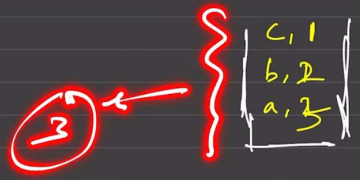
4

Longest substring with at most K distinct characters

s = a a b b c c d k = 2

Longest substring with at most K distinct characters

s = a a a b b c c d k = 2



```
for ( i = 0 → n )
{
    mpp.clear()

    for ( j = i → n)
    {
        mpp[ s[j] ]++;

        if ( mpp.size() <= 1 )
            maxlen = max( maxlen, j - i + 1 );
    }
```

```
for ( i = 0 → n )
{
    mpp.clear()

    for ( j = i → n )
    {
        mpp[ s[j] ]++;

        if ( mpp.size() <= 1 )
            maxLen = max( maxLen, j - i + 1 );
        else
            break;
    }
}
return maxLen;
}
```

Longest substring with atmost K distinct characters

256

s = a a a b b c c d k = 2

fun (s, k)

{

maxlen = 0 , mpp <char, int>

fun (i = 0 → n)

{

mpp.clear()

fun (j = i → n)

{

mpp[s[j]]++;

if (mpp.size() <= k)

TC → O(N²) × log(26)

SC → O(256)

Longest substring with at most K distinct characters

s = a a a b b c c d k = 2

i j

c	1
b	2
a	3

maxlen = 0 1 2 3 4 5

Longest substring with at most K distinct characters

$s = a\ a\ a\ b\ b\ c\ d \qquad k = 2$

c	1
b	2
a	2

maxlen = ~~0 1 2 3 4~~ 5

3

Longest substring with at most K distinct characters

s = a a a b b c d k = 2

Longest substring with at most K distinct characters

s = a a a b b c c d k = 2

l r

c	2
b	2
d	1

maxlen = 5

s = a a a b b c c d k = 2

l r

```
fun( s, k)
{
    maxlen = 0, l = 0, r = 0, mpp < char, int >

    while (r < s.size())
    {
        mpp[s[r]]++;
```

```
while ( mpp.size() > k )
{
        hash[s[l]]--;
        if (hash[s[l]]==0) mpp.erase(mpp[s[l]]);
          l = l+1;
}
if (
```

```
func( s, k )
{
    maxlen = 0 , l = 0, r = 0, mpp < char , int >

    while ( r < s.size() )
    {
        mpp[ s[r] ]++ ;

        while ( mpp.size() > k )
        {
            hash[s[l]]--;
            if (hash[s[l]] == 0) mpp.erase (mpp[s[l]]);
                l = l + 1;
        }
```

```
if (mpp.size() <= k)
    maxlen = max(maxlen, r-l+1);

r = r+1;
}
return maxlen;
```

s = a a a b b c c d k = 2

l r

TC → O(N) + O(N) + O

```
func( s, k)
{
    maxlen = 0, l = 0, r = 0, mpp < char, int >

    while (r < s.size())
    {
        mpp[s[r]]++;

        while (mpp.size() > k)
        {
            hash[s[l]]--;
            if (hash[s[l]] == 0) mpp.erase(mpp[s[l]]);
            l = l + 1;
        }
    if (mpp.size() <= k)
```

Longest substring with at most K distinct characters

s = a a a b b c c d K = 2

l r

TC → O(N) + O(N) + O(log 256)

SL → O(256)

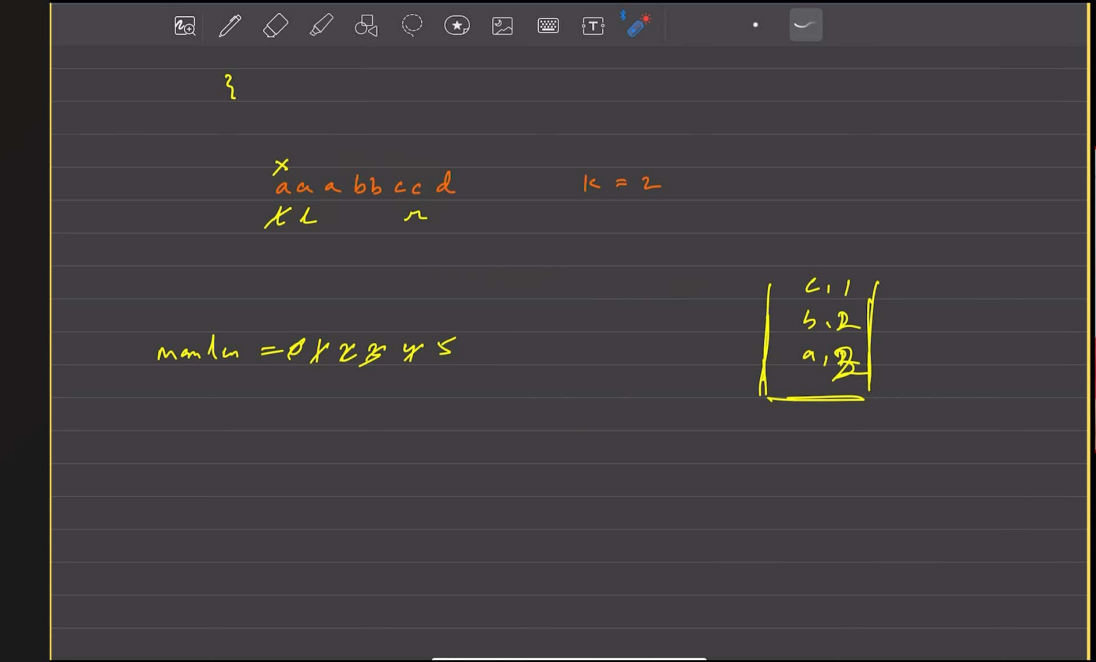
```
func( s, k)
{
    maxLen = 0, l = 0, r = 0, mpp < char, int >

    while (r < s.size())
    {
        mpp[s[r]]++;

        while ( mpp.size() > k)
        {
            hash[s[l]]--;
            if (hash[s[l]] == 0) mpp.erase(mpp[s[l]]);
```

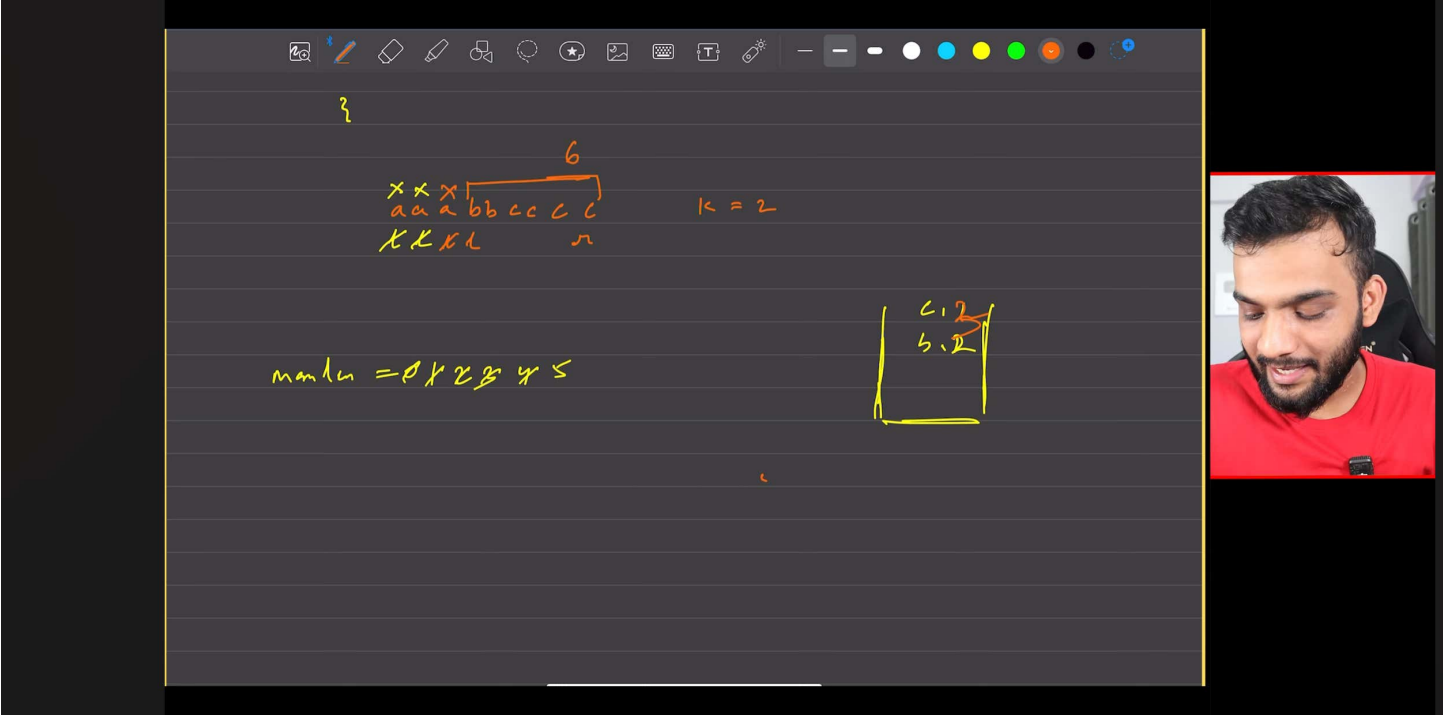
}

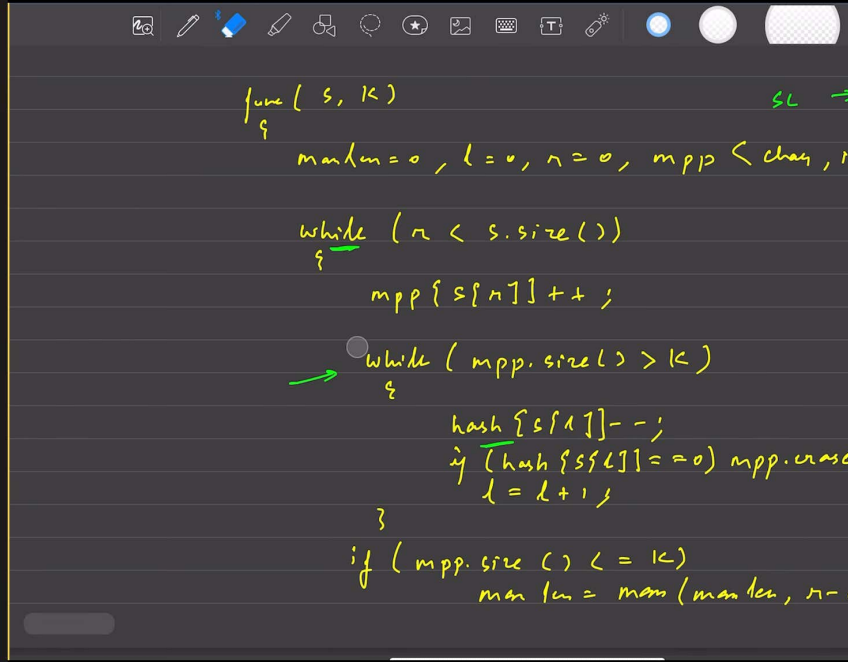
aaabbcd k = 2

c : 1
b : 2
a : 3

maxLen = 0 1 2 3 4 5


aa a bb cc d
k = 2
c:1
b:2
a:2


}
6
k = 2
c : 2
b : 2
maxLen = 0 1 2 3 4 5


+ O(log 256)
SL → O(256)
func (s, k)
{
maxLen = 0, l = 0, r = 0, mpp < char, int >
while (r < s.size())
{
mpp[s[r]]++;
while (mpp.size() > k)
{
hash[s[l]]--;
if (hash[s[l]] == 0) mpp.erase(mpp[s[l]]);
l = l + 1;
}
if (mpp.size() <= k)
maxLen = max(maxLen, r-l+1);

+ O(log 256)

SL → O(256)

```
func( s, k)
{
    maxlen = 0, l = 0, r = 0, mpp < char, int >

    while (r < s.size())
    {
        mpp[s[r]]++;

        if (mpp.size() > k)
        {
            hash[s[l]]--;
            if (hash[s[l]] == 0) mpp.erase(mpp[s[l]]);
            l = l + 1;
        }
        if (mpp.size() <= k)
            maxlen = max(maxlen, r - l + 1);
```

```
maxLen = 0, l = 0, r = 0, mpp<char, int>

while (r < s.size())
{
    mpp[s[r]]++;

    if (mpp.size() > k)
    {
        hash[s[l]]--;
        if (hash[s[l]] == 0) mpp.erase(mpp[s[l]]);
        l = l + 1;
    }

    if (mpp.size() <= k)
        maxLen = max(maxLen, r - l + 1);

    r = r + 1;
}
```

Longest substring with at most K distinct characters

s = a a a b b c c d k = 2

l r

TC → O(N) + O(log 256)

SL → O(256)

```
func( s, k)
{
    maxLen = 0, l = 0, r = 0, mpp < char, int >

    while (r < s.size())
    {
        mpp[s[r]]++;

        if (mpp.size() > k)
        {
            hash[s[l]]--;
            if (hash[s[l]] == 0) mpp.erase(mpp[s[l]]);
```