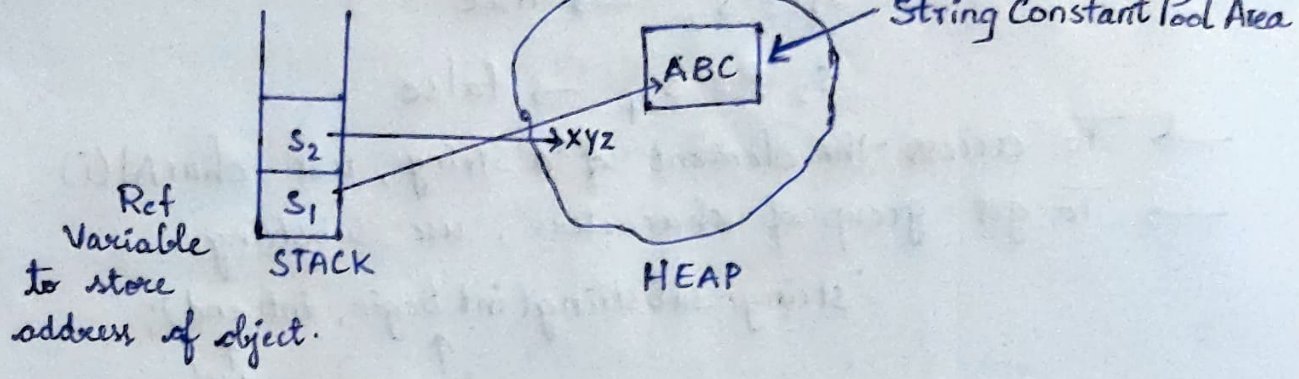
Strings:

* String is a sequence of individual characters.

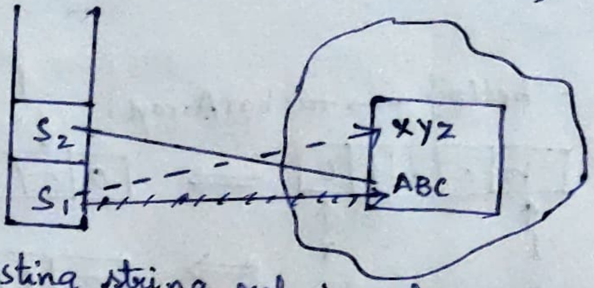
String s_1 = "ABC"; → literal

String s_2 = new String ("xyz") ← object.


String Constant Pool Area
ABC
xyz
s_2
s_1
Ref Variable to store address of object.
STACK
HEAP

→ Why Strings are Immutable?

String s_1 = "ABC", s_1 = "xyz"; String s_2 = "ABC"



→ when the existing string ref is changed, it will not change the existing object, and it will create a new literal and s_1 points to that new literal.

→ If we create a new String literal s_2 with ABC, As ABC literal is available already s_2 points to ABC.

→ == and . equals:-

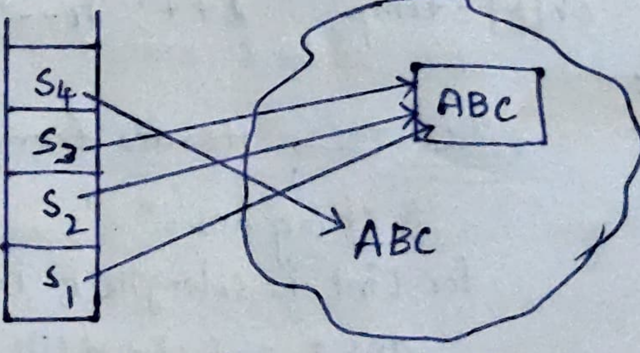
String s_1 = "ABC";

s_2 = "ABC";

$s_1 == s_2$

↓

True → Because s_1 and s_2 points to the same Address.



== operator for Add. Comparison

.equals for Value Comparison.

String s3 = "ABC";   s4 = new String ("ABC");

s3 == s4 → false

s3 . equals (s4) → True

s1 == s3 → True

s3 == s2 → True

s2 == s4 → false

→ To access the element of a String, use charAt(i)

→ To get group of characters, use substring.

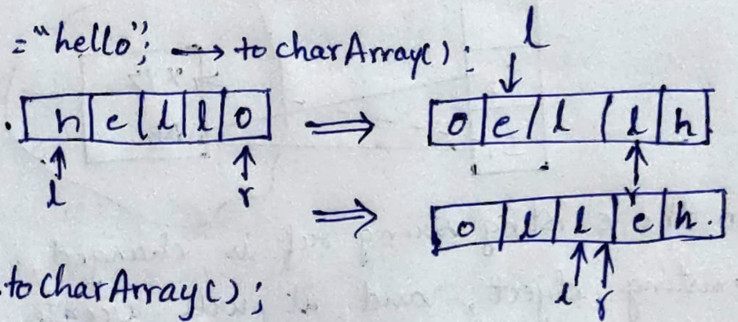
"string". substring(int begin, int end);

begin ↑ inclusive   end ↑ exclusive

"string". substring (0, 3) → o/p: str

Reverse a String:

String s = "hello"; → to charArray():



```
char[] ch = s.toCharArray();
int l = 0;    int r = s.length() -1;
while (l < r)
{
    char temp = ch[l];
    ch[l] = ch[r];
    ch[r] = temp;    l++; r--;
}
```

(or) → iterate from right

```
String ans = " ";
for (int i = s.length()-1; i>=0; i--){
    ans + = s.charAt(i);
}
```

Reverse a String Using Substring:

```
String s= "hello world" → length = 11
i = 0 ;   a = s.substring (1, s.length() - i)
            = s.substring(1, 11-0) ⇒ s.substring(1, 11)
            = ello world
          b = s.charAt(0) → h.
          c = s.substring (s.length()-i)
            = s.substring (11-0) = ""
After Step ⓪:  s = ello worldh.
i = 1 ;   a = s.substring (1, 11-1)
            = llo world
          b = s.charAt(0) → e
          c = s.substring(11-1) → h.
After Step ①:  s = llo worldeh.
i = 2 ;   a = s.substring (1, 11-2)
            = lo world
          b = l      c = s.substring(11-2) → eh.
After step ②:  s = lo worldleh.
        :
        :
After step ⑩:  s = dlrow olleh.

for (int i=0; i<s.length; i++)
{
  String a = s.substring(1, s.length() - i);
         b = s.charAt(0);
         c = s.charAt(s.length() - i);
         s = a+b+c;
}
return s;
```

Longest SubString without repeating Characters:

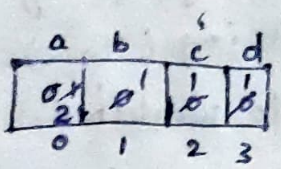
→ abcdacbde
→ abcd 4
→ bcda 4
→ acbde $\boxed{5}$ ✓

This is Variable Type of Sliding Window

→ **Brute Force:**

abcda ...

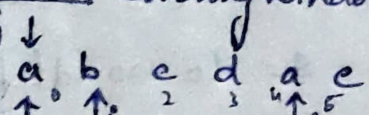
1. Find / calculate all the substrings for each character



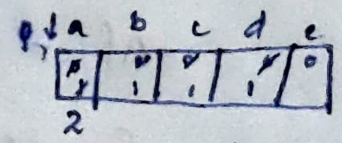
→ If any character has a value greater than one
⇓
we have duplicate instances.

```
for (int i = 0; i < s.length(); i++)
{
    for (int j = i+1; j < s.length(); j++) {
        // all unique values are present,
           if yes update the window length.
    }
}
```

→ **Optimized Approach:** Sliding Window



int i = 0, j = 0,
max = -∞
int [] SArr = new int [26];

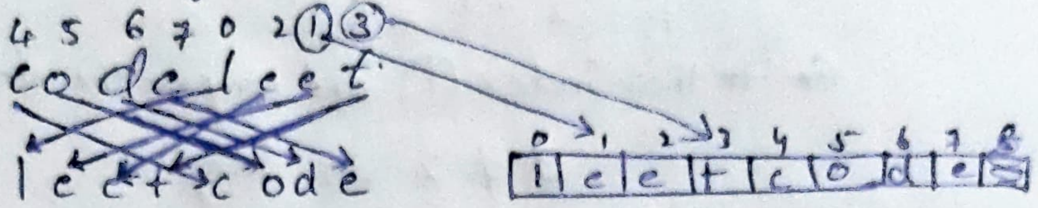


j - i + 1 = 4 - 0 + 1
= 5
i = ①

```
while (j < s.length()){
    int index = s.charAt(j) - 'a';
    SArr[index]++;
    while ( SArr[index] > 1)
    {
        SArr[ ~~inde~~ s.charAt(i) - 'a']--;
        i++;
    }
    max = Math.max( max, j-i+1);
    j++;
}
return max;
```

Shuffle the String:

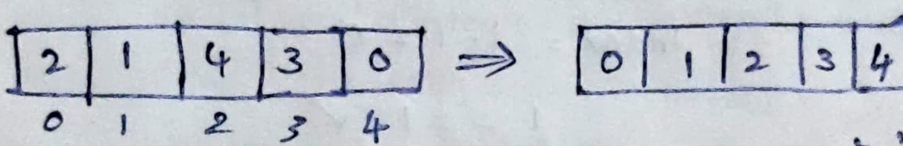
You are given an array indices of same length. The string s will be shuffled such that the character at ith position moves to the indices[i] in the shuffled string.



i.e get the index of the character it should fit and the character and place it in the new array.

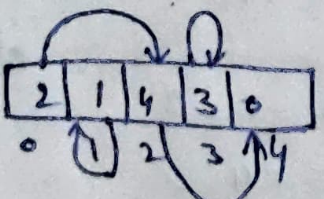
```
for (int i=0; i<s.length(); i++){
    SArr[ indices[i] ] = s.charAt(i);
}
```

Cycle Sort:

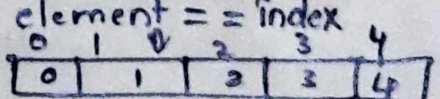


→ If the array elements are in the range 0-n then we can sort O(n).

→ Cycle Sort: Sort the elements based on the index of array



i.e element == index



If the elements are between 1 to 'n' then

element = index+1

Note :-

If the element is at its correct location keep it as it is, if not swap it.

0	1	2	3	4
2	3	1	4	5

$\Rightarrow$ (1, n).

element = 2
index = element − 1
= 2 − 1 = 1

Go to that index ① and compare the element.

3 ≠ 2 ⇒ Swap

0	1	2	3	4
③	2	1	4	5

element = 3
index = 3 − 1 = ②

1 ≠ 2 ⇒ Swap

0	1	2	3	4
1	2	3	4	5

element = ①
index = 1 − 1 = 0

1 == 1 ✓

next i++;

element = 2
index = 2 − 1 = 1
2 == 2 ✓
i++

element = 3
index = 3 − 1 = 2
3 == 3 ✓
i++ ✓

element = 4
index = 4 − 1 = 3
4 == 4 ✓
i++

element = 5
index = 5 − 1 = 4
5 = 5 ✓

```
int i = 0;
while(i < arr.length){
    element = arr[i];
    index = element-1;
    if (arr[i] < arr.length && arr[index] != element)
        swap(arr, i, index);
    }
else{ i++;
}}
```

Shuffle the String Using Cycle Sort:

```
indices → 4  5  6  7  0  2  1  3

          c  o  d  e  l  e  e  t
          0  1  2  3  4  5  6  7
```

a) element = c
   index = 4
   i ⇒ 0
   0! = 4
   Swap

   element = 'o'
   index = 5
   i = 1
   5! = 1
   Swap

```
char[] ch = s.toCharArray();
int i=0;
while(i < indices.length()) {
    char ch = ch[i];
    int index = indices[i];
    if(i! = index) {
        char temp = ch[i];
        ch [i] = ch [index];
        ch[index] = temp;            } Swapthe element.

        int tempIndex = indices[i];
        indices [i] = indices [index];
        indices [index] = tempIndex;  } Swap the indices
    }
    else {
        i++;
    }
}
```

→ Find all Anagrams in a String:

$s = "\overset{0}{a}\overset{1}{b}\overset{2}{c}\overset{3}{b}\overset{4}{c}\overset{5}{a}\overset{6}{d}"$ o/p : 2

$p = "abc"$ o/p:- 0, 3

i.e expecting the window start of the anagram.

→ Count frequency of all letters in a String.
String contains only lowercase letters.

I/P:- s = "abcabcd"

O/P:- a → 2, b → 2, c → 2, d → 1

①: calculate the frequency of each character.

② get the frequency of each character & append the value to the string and set the frequency of that character to zero to avoid duplicates.

```
int[] array = new int [26];
for (char c : ~~array~~s.toCharArray()){
        array [c - 'a']++;
    }

StringBuilder sb = new StringBuilder();
```

```
for(int i=0; i<s.length; i++)
{
  char current = s.charAt(i);
  if (array[current-'a'] >0)
   {
     sb.append(current);
     sb.append(array[current-'a']);
     array [current-'a']=0;
}}
```

→ First Unique Character in String.

Input: "leetcode" o/p: 0

Input: "loveleetcode" o/p:- 2
012

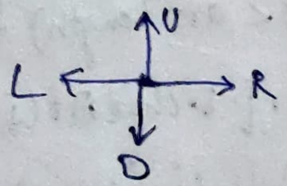
→ ①:- Store the frequency of each character in an array.

②:- get the frequency of each character of string & check if the frequency is 1. , if yes return the value's index.

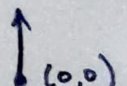
```
int [] array = new int [26];
for (char c: s.toCharArray()){
    array[~~s.ch~~ c-'a']++;
}

    for (int i = 0; i < s.length; i++)
    {
        if ( array[ s.charAt(i)-'a'] == 1) {
            return i;
        }
    }
    return -1;
```

→ Robot return to origin

There is a robot starting at the position (0,0), the origin, on 2D plane. Given a sequence of its moves, judge if this ends up at [0, 0]. after completes its moves.



I/P: UD



O/P: True

I/P: LL

O/P: false

Approach:

Take 4 Counters, CounterU, ConterD, Counter L, CounterR

If Counter U == Conunter D } return true
L == R   else false,

Longest Repeating Character Replacement:

S = "ABAB", K = 2 o/p: 4

→ Replace the two A's with two B's (or) vice-versa.

S = "AABABBA", K = 1 o/p: 4

Replace the one 'A' with B & form AA $\underbrace{\text{BBBB}}_{\boxed{4}}$A

S = ABAB → Replace B with A

$\underline{\text{AAAA}} \rightarrow \boxed{4}$

S = $\underline{\text{ABCAAA}}$ → K = 2

Replace B & C with A.

AAAAAA → $\underline{\text{o/p: 6}}$

A	4
B	1
C	1

length of string / window = 6

count of left overs = window size - max

= 6 - 4

= 2

```
0 1 2 3
A B C A A A
↑   ↑
i   j

2-0+1 > k

3 > 2
```

```
while (j < arr.length) {
    SArr[ s.charAt(j) - 'a']++;
    while (j-i+1 - Max > k)
    {
        SArr [S.charAt (i)]--;
        i++;
    }
    max = Max (max, j-i+1);
    j++;
}
```