

# Strings

## Introduction

The image shows a video frame from a lecture. A man in a blue polo shirt and glasses is gesturing with his right hand towards the right side of the frame. He appears to be speaking. In the background, there is a whiteboard with handwritten notes. At the top left of the whiteboard, it says "122015115@saitra.ac.in". At the top right, it says "GeeksforGeeks" and "A computer science portal for geeks". Below this, the title "String Introduction" is written. To the left of the title, there are three points: "⇒ Sequence of characters", "⇒ Small Set", and "⇒ Contiguous integer values for 'a' to 'z' and 'A' to 'Z' in both ASCII and UTF-16". There is a handwritten note "65" next to "UTF-16". To the right of the title, there is a diagram comparing C/C++ and Java. It shows "C/C++" with "char : ASCII 8 Bit" and "Also supports wchar\_t". It shows "Java" with "UTF-16 16 Bit" and "Also supports byte". Above the Java section, there is a handwritten note "97 'b' > 96". At the bottom of the whiteboard, there is a watermark that says "Activate Windows" and "www.microsoft.com/activate/windows". The video player interface at the bottom shows a progress bar at 2:57, a volume icon, and other standard video controls.

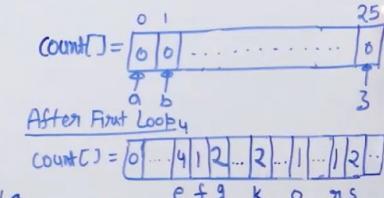
- Strings are collections of characters.... For C++, 8Bit ASCII and for Java, 16Bit UTF-16 code. Both ASCII and UTF-16 have the same encoding for the first 128 characters. For storing additional languages UTF-16 has another 8 bits extra

## String - Introduction

Example Problem: Print Frequencies of character (in sorted order) in a string of lower case alphabets.

**C++**      **O/P:**

```
String str = "geeksforgEEKS";
int count[26] = {0};
for (int i=0; i<str.length(); i++)
    count[str[i] - 'a']++;
for (int i=0; i<26; i++)
    if (count[i] > 0)
        cout << (char)(i + 'a') << " "
            << count[i] << endl;
```



### Java

```
String str = "geeksforgEEKS";
int []count = new int[26];
for (int i=0; i<str.length(); i++)
    count[str.charAt(i) - 'a']++;
for (int i=0; i<26; i++)
    if (count[i] > 0)
        System.out.println((char)(i + 'a') +
                           " " + (count[i]).
```

Activate Windows  
Go to Settings to activate Windows.

## Strings in C++

- Two ways of storing strings
  - C style : using array of characters
  - C++ style : using in-built string class

## Strings in C++

### ① C-Style Strings

```
int main()
{
    char str[] = "gfg";
    cout << str;
    return 0;
}
```

Activate Windows  
Go to Settings to activate Windows.

## Strings in C++

### ① C-Style Strings

{'g', 'f', 'g', '\0', '\0', '\0', '\0'}

```
int main()
{
    char str[] = "gfg";
    cout << str;
    return 0;
}
```

Activate Windows  
Go to Settings to activate Windows.

- If we initialize with more size, then all other cells will be filled with '\0' char.

## Strings in C++

### ① C-Style Strings

```
int main()
{
    char str[] = "gfg";
    cout << sizeof(str);
    return 0;
}
```

Activate Windows  
Go to Settings to activate Windows.

- Here we get 4 as output considering the size of a char is 1 byte becz size of the char array here also includes the '\0' character.

Strings in C++

C-Style Strings

```
int main()
{
    char str[] = {'g', 'f', 'g'};
    cout << str;
    return 0;
}
```

gfg

Activate Windows  
Go to Settings to activate Windows.

- Prints gfg along with some random characters or segmentation fault or gfg alone becz when initializing as a char array we need to put '\0' explicitly. If we initialize as char str[] = "gfg", the compiler automatically puts '\0' at the end

a  
aa  
a...  
ab  
ab...  
ac

Strings in C++

① C-Style Strings

```
int main()
{
    char s1[] = "abd";
    char s2[] = "bcd";
    int res = strcmp(s1, s2);
    if (res > 0)
        cout << "Greater";
    else if (res == 0)
        cout << "Same";
    else
        cout << "Smaller";
}
```

Functions

strlen(str)  
strcmp(s1, s2)  
strcpy(s1, s2)

Activate Windows  
Go to Settings to activate Windows.

- strlen(str) gives length of the string excluding '\0' char.

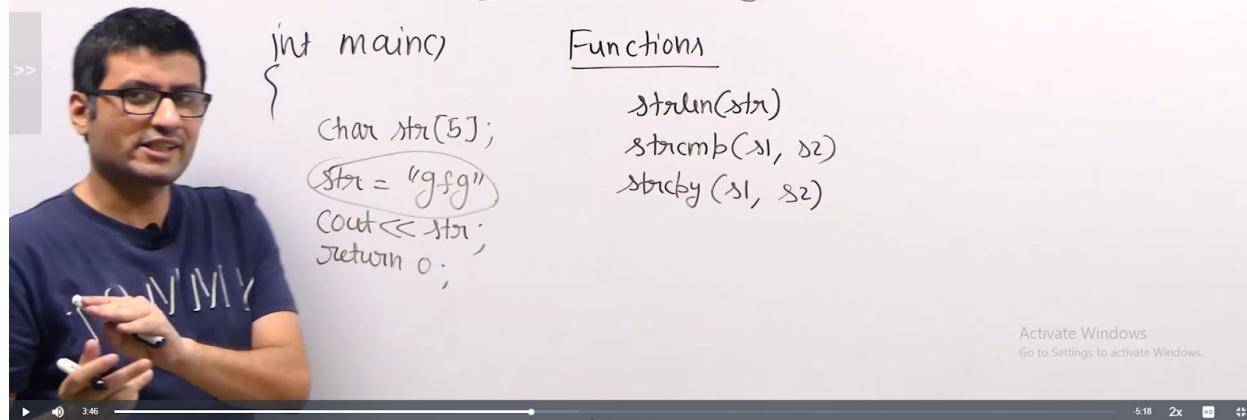
- `strcmp(s1, s2)` compares two strings lexicographically. If `s1` is greater than `s2` lexicographically then it returns +ve val or if smaller returns -ve val or if it is the same then returns 0.

122015115@safra.ac.in

**GeeksforGeeks**  
A computer science portal for geeks

Strings in C++

① C-Style Strings



Activate Windows  
Go to Settings to activate Windows.

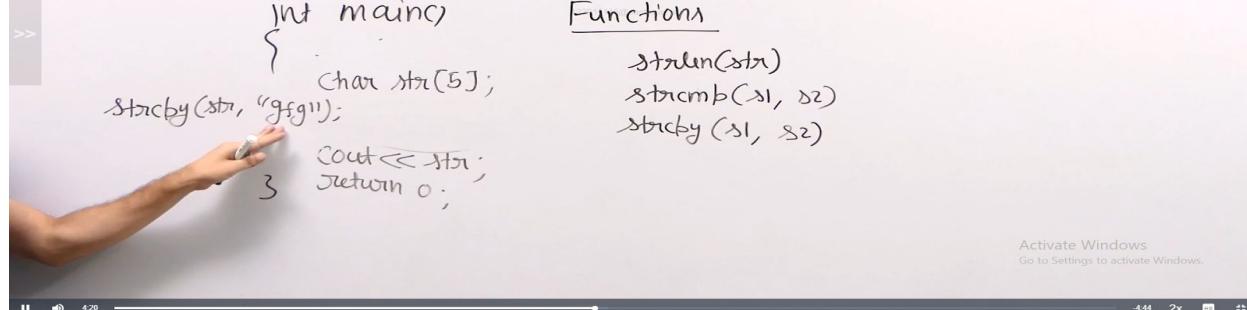
3:46 5:18 2x

- The program will generate compiler error because `str` is a char array, `str` gives the address of the array and thus it can't be used in the left hand side of the assignment statement. So here we need `strcpy()`

**GeeksforGeeks**  
A computer science portal for geeks

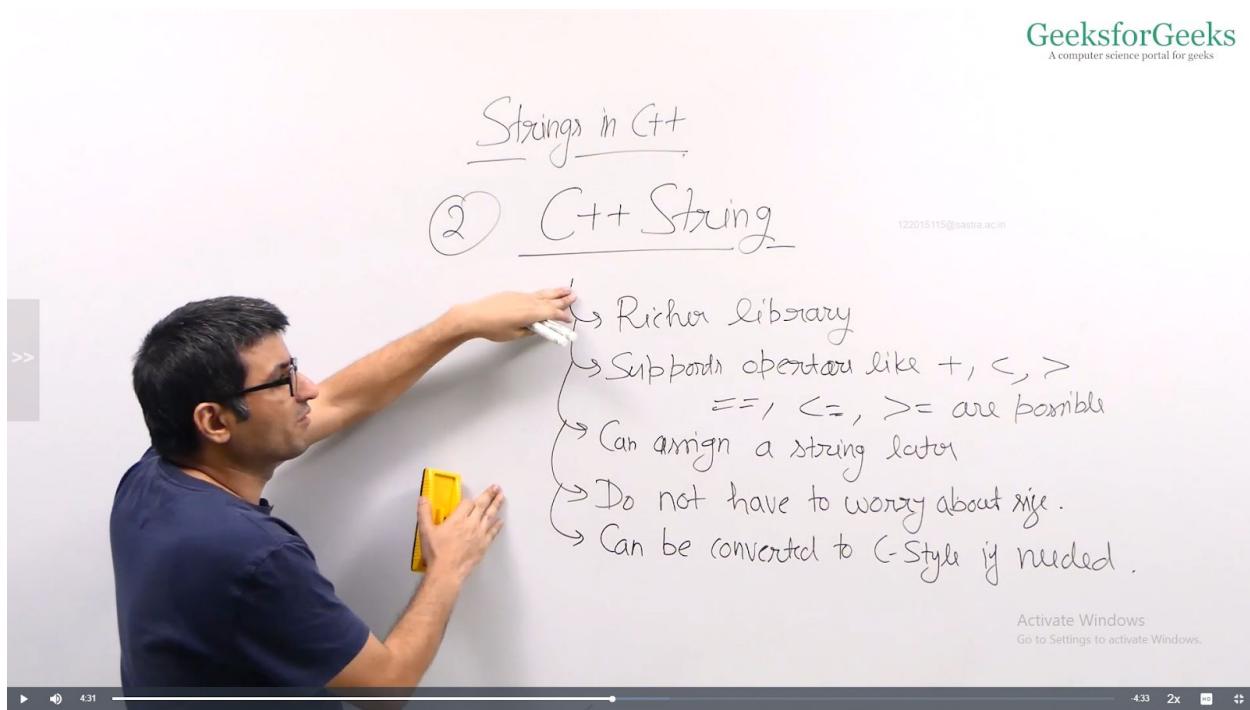
Strings in C++

① C-Style Strings

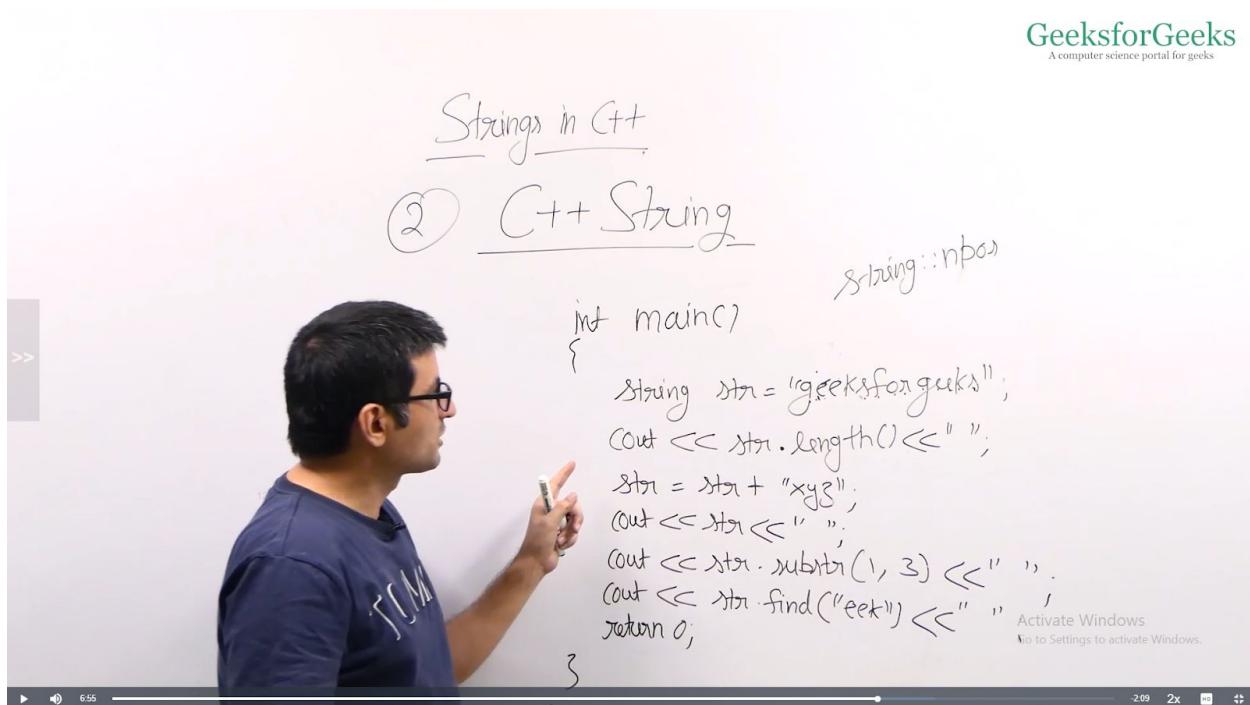


Activate Windows  
Go to Settings to activate Windows.

4:20 4:44 2x



- Easier to use C++ style strings
- String is in-built class in C++

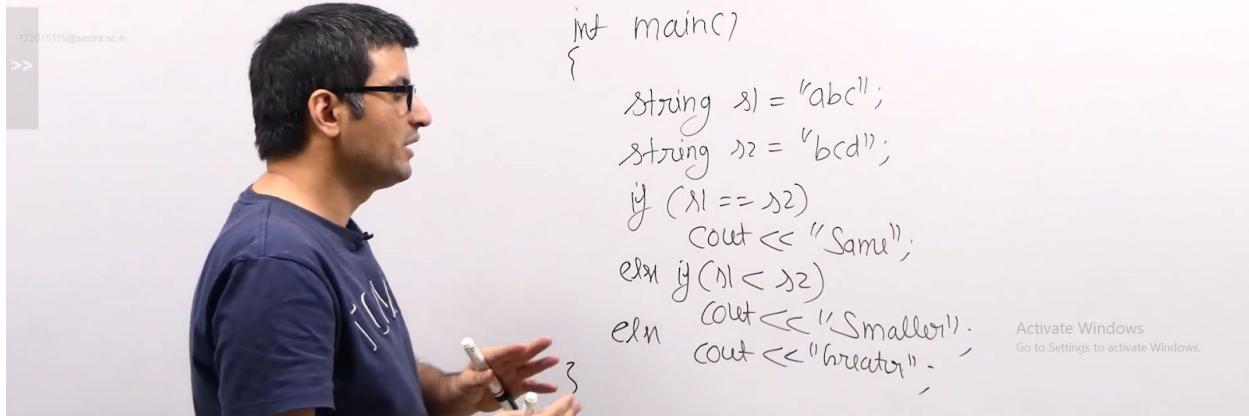


Output: 13 geeksforgeeksxyz eek 1

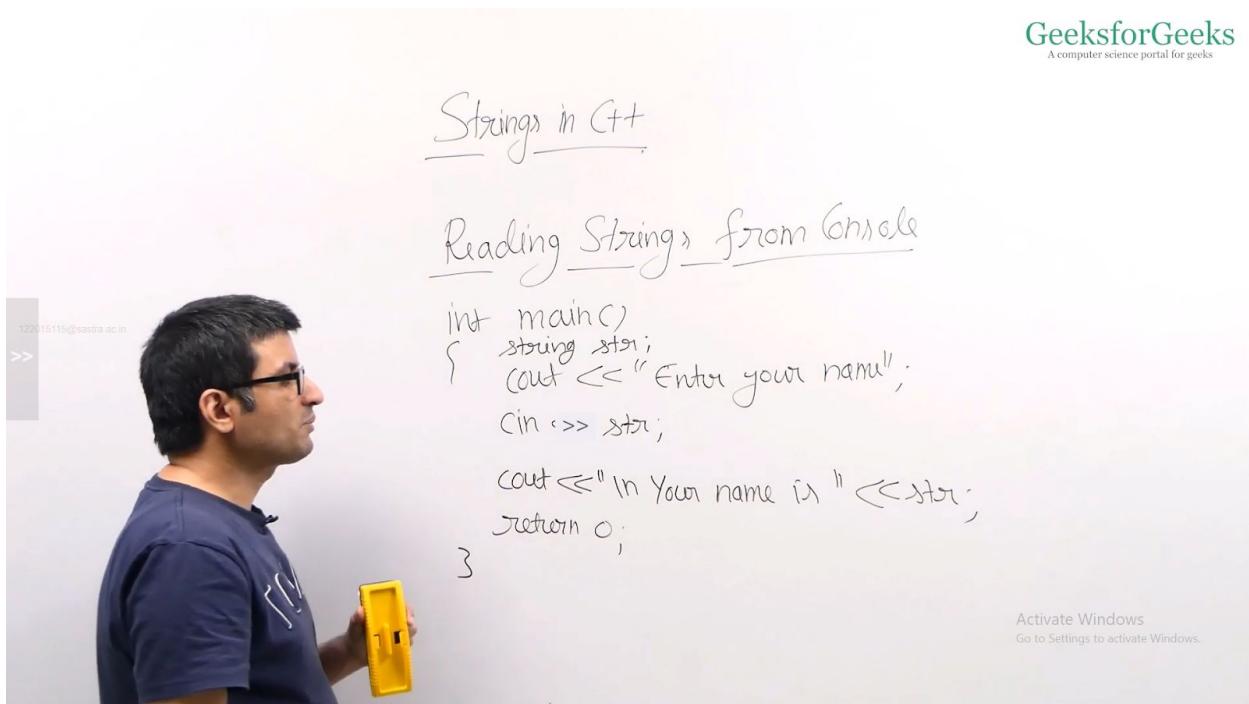
- str.find(str1) gives index of str1 in str. If not present it will return string::npos
- str.find(str1, pos) gives index of str1 in str after pos if it is present(same as java)

## Strings in C++

### ② C++ String



- Usage of comparison operators



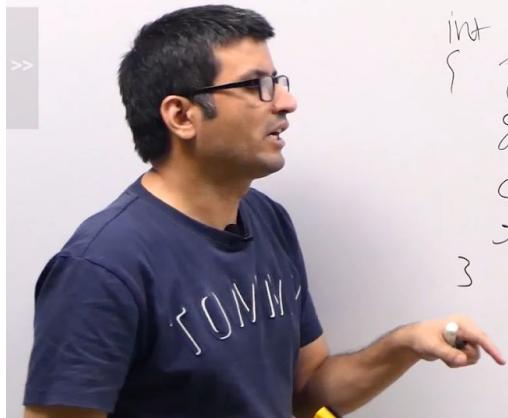
- This accepts strings that have only one word, i.e., it will not stop taking input if space or '\n' is given. In order to use those, getline() is used

## Strings in C++

### Reading String from Console

```
int main()
{
    string str;
    cout << "Enter your name";
    getline(cin, str);

    cout << "Your name is " << str;
    return 0;
}
```



Activate Windows  
Go to Settings to activate Windows.

- This will take input till enter is pressed
- `getline(cin, str, '$')` will stop taking input if we press '\$' only. By default, the 3rd parameter will be '\n'

## Strings in C++

### Iterating through a string

```
int main()
{
    string str = "geeksforgeeks";
    for(int i=0; i<str.length(); i++)
        cout << str[i];
    cout << endl;
    for(char x: str)
        cout << x;
```



Activate Windows  
Go to Settings to activate Windows.

- Traversing a string using for and foreach loop

## Check for Anagram

Given two strings, we need to check if these strings are Anagram of each other or not.

122015115@sastra.ac.in

GeeksforGeeks  
A computer science portal for geeks

Check for Anagram

I/p :  $s_1 = \text{"listen"}$ ,  $s_2 = \text{"silent"}$   
O/P : Yes

I/p :  $s_1 = \text{"aaacb"}$ ,  $s_2 = \text{"cabaa"}$   
O/P : Yes

I/p :  $s_1 = \text{"aab"}$ ,  $s_2 = \text{"bab"}$   
O/P : No

Activate Windows  
Go to Settings to activate Windows.

122015115@sastra.ac.in

GeeksforGeeks  
A computer science portal for geeks

Naive Solution

C++

```
bool areAnagram(String s1, String s2){  
    if (s1.length() != s2.length())  
        return false;  
    Sort(s1.begin(), s2.end());  
    Sort(s2.begin(), s2.end());  
    return (s1 == s2);  
}
```

Java

```
boolean areAnagram(String s1, String s2){  
    if (s1.length() != s2.length())  
        return false;  
    Sort First String  
    char a1[] = s1.toCharArray();  
    Arrays.sort(a1);  
    s1 = new String(a1);  
    Sort Second String  
    char a2[] = s2.toCharArray();  
    Arrays.sort(a2);  
    s2 = new String(a2);  
    return s1.equals(s2);  
}
```

Activate Windows  
Go to Settings to activate Windows.

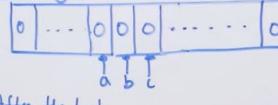
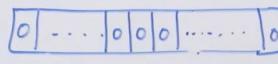
- Time Complexity :  $O(n \log n)$
- Space Complexity:  $O(1)$

```

const int CHAR = 256;
bool areAnagram(String s1, String s2)
{
    if(s1.length() != s2.length())
        return false;
    int count[CHAR] = {0};
    for(int i=0; i<s1.length(); i++)
    {
        count[s1[i]]++;
        count[s2[i]]--;
    }
    for(int i=0; i<CHAR; i++)
        if(count[i] != 0)
            return false;
    return true;
}

```

### Efficient Solution

s1 = "aabca"  
 s2 = "acaba"  
 Initially:  
  
 After the loop:  


```

static final int CHAR = 256;
boolean areAnagram(String s1, String s2)
{
    if(s1.length() != s2.length())
        return false;
    int count[] = new int[CHAR];
    for(int i=0; i<s1.length(); i++)
    {
        count[s1.charAt(i)]++;
        count[s2.charAt(i)]--;
    }
    for(int i=0; i<CHAR; i++)
        if(count[i] != 0)
            return false;
    return true;
}

```

Activate Windows  
Go to Settings to activate.

- Time : O(n + CHAR)
- Space : O(CHAR)

## LeftMost Repeating Character

Given a string, the task is to find the first character (whose leftmost appearance is first) that repeats.

Leftmost Repeating Character  
 I/p : str = "geeksforgeeks"  
 O/p : 0

I/p : str = "abbcc"

O/p : 1

I/p : str = "abcd"

O/p : -1

Activate Windows  
Go to Settings to activate Windows.

## Naive Approach

```
C++ >> int leftMost(String str)
{
    for (int i=0; i<str.length(); i++)
    {
        for (int j=i+1; j<str.length(); j++)
        {
            if (str[i] == str[j])
                return i;
        }
    }
    return -1;
}
```

"cabbad"

```
int leftMost(String str)
{
    for (int i=0; i<str.length(); i++)
    {
        for (int j=i+1; j<str.length(); j++)
        {
            if (str[i] == str[j])
                return i;
        }
    }
    return -1;
}
```

Java

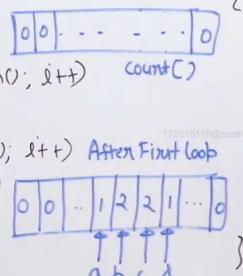
122015115@sastra.ac.in

Activate Windows  
Go to Settings to activate Windows 10

- Time:  $O(n^2)$
- Space:  $O(1)$

## Better Approach

```
const int CHAR = 256;
int leftMost(String str)
{
    int count[CHAR] = {0};
    for (int i=0; i<str.length(); i++)
        count[str[i]]++;
    for (int i=0; i<str.length(); i++)
        if (count[str[i]] > 1)
            return i;
    return -1;
}
```



```
static final int CHAR = 256;
int leftMost(String str) Java
{
    int [] count = new int[CHAR];
    for (int i=0; i<str.length(); i++)
        count[str.charAt(i)]++;
    for (int i=0; i<str.length(); i++)
        if (count[str.charAt(i)] > 1)
            return i;
    return -1;
}
```

Activate Windows  
Go to Settings to activate Windows 10

Efficient Approach - 1

```

const int CHAR = 256;
int leftmost(string str)
{
    int fIndex[CHAR];
    fill(fIndex, fIndex + CHAR, -1);
    int res = INT_MAX;
    for (int i = 0; i < str.length(); i++)
    {
        int fi = fIndex[str[i]];
        if (fi == -1)
            fIndex[str[i]] = i;
        else
            res = min(res, fi);
    }
    return (res == INT_MAX) ? -1 : res;
}

```

Java

```

static final int CHAR = 256;
int leftmost(String str)
{
    int fIndex[] = new int[CHAR];
    Arrays.fill(fIndex, -1);
    int res = Integer.MAX_VALUE;
    for (int i = 0; i < str.length())
    {
        int fi = fIndex[str.charAt(i)];
        if (fi == -1)
            fIndex[str.charAt(i)] = i;
        else
            res = Math.min(res, fi);
    }
    return (res == Integer.MAX_VALUE) ? -1 : res;
}

```

Efficient Approach - 2

```

const int CHAR = 256;
int leftmost(string str)
{
    bool visited[CHAR];
    fill(visited, visited + CHAR, false);
    int res = -1;
    for (int i = str.length() - 1; i >= 0; i--)
    {
        if (visited[str[i]])
            res = i;
        else
            visited[str[i]] = true;
    }
    return res;
}

```

Java

```

static final int CHAR = 256;
int leftmost(String str)
{
    boolean visited[] = new boolean[CHAR];
    int res = -1;
    for (int i = str.length() - 1; i >= 0; i--)
    {
        if (visited[str.charAt(i)])
            res = i;
        else
            visited[str.charAt(i)] = true;
    }
    return res;
}

```

- Time: O(n) and Space: O(CHAR). 1st approach has more no of comparisons than 2nd

## LeftMost Non-Repeating Element

Given a string, the task is to find the leftmost character that does not repeat.

GeeksforGeeks  
A computer science portal for geeks

122015115@sasstra.ac.in

Leftmost Non-Repeating Character

I/p: str = "geeksforgeeks"  
O/p: 5 //Index of 'f'

>>

I/p: str = "abcabc"  
O/p: -1

I/p: str = "apple"  
O/p: 0 //Index of 'a'

Activate Windows  
Go to Settings to activate Windows.

GeeksforGeeks  
A computer science portal for geeks

122015115@sasstra.ac.in

Naive Solution

int nonRep(string str){  
 for(int i=0; i<str.length(); i++)  
 {  
 bool flag = false;  
 for(int j=0; j<str.length(); j++)  
 {  
 if(i!=j && str[i]==str[j])  
 flag = true;  
 }  
 if(flag == false)  
 return i;  
 }  
 return -1;  
}

(++)

geeksforgeeks  
 $O(n^2)$

Java

Activate Windows  
Go to Settings to activate Windows.

Better Solution  
(Two Traversals)

```

const int CHAR = 256;
int nonRep(String str)
{
    int count[CHAR] = {0};
    for(int i=0; i<str.length(); i++)
        count[str[i]]++;
    for(int i=0; i<str.length(); i++)
        if(count[str[i]] == 1)
            return i;
    return -1;
}

```

Java

```

Static final int CHAR = 256;
int nonRep(String str)
{
    int []count = new int[CHAR];
    for(int i=0; i<str.length(); i++)
        count[str.charAt(i)]++;
    for(int i=0; i<str.length(); i++)
        if(count[str.charAt(i)] == 1)
            return i;
    return -1;
}

```

C++

After First Traversal

Activate Windows  
Go to Settings to activate Windows.

Efficient Solution  
(One Traversal)

```

const int CHAR = 256;
int nonRep(String str)
{
    int fI[CHAR];
    fill(fI, fI+CHAR, -1);
    for(int i=0; i<str.length(); i++)
    {
        if(fI[str[i]] == -1)
            fI[str[i]] = i;
        else
            fI[str[i]] = -2;
    }
    int mI = Integer.MAX_VALUE;
    for(int i=0; i<CHAR; i++)
        if(fI[i] >= 0)
            mI = min(mI, fI[i]);
    return (mI == Integer.MAX_VALUE) ? -1 : mI;
}

```

Java

```

Static final int CHAR = 256;
int nonRep(String str)
{
    int []fI = new int[CHAR];
    Arrays.fill(fI, -1);
    for(int i=0; i<str.length(); i++)
    {
        if(fI[str.charAt(i)] == -1)
            fI[str.charAt(i)] = i;
        else
            fI[str.charAt(i)] = -2;
    }
    int mI = Integer.MAX_VALUE;
    for(int i=0; i<CHAR; i++)
        if(fI[i] >= 0)
            mI = min(mI, fI[i]);
    return (mI == Integer.MAX_VALUE) ? -1 : mI;
}

```

Activate Windows  
Go to Settings to activate Windows.

## Reverse Words in a String

Given a string, reverse its words.

122015115@sastrac.ac.in

Reverse Words in a String

I/p : str = "Welcome to gfg"  
O/p : str = "gfg to Welcome"

I/p : str = "I love coding"  
O/p : str = "coding love I"

I/p : str = "abc"  
O/p : str = "abc"

Activate Windows  
Go to Settings to activate Windows.

GeeksforGeeks  
A computer science portal for geeks

122015115@sastrac.ac.in

Reverse Words in a String

I/p : str = "Welcome to gfg"  
O/p : str = "gfg to Welcome"

Naive Method

① Create a Stack  
② Push words one by one to the stack  
③ Pop words from the stack and append to output.

Stack Diagram:  
gfg  
to  
Welcome

str = ""  
str = "gfg to Welcome!"

Activate Windows  
Go to Settings to activate Windows.

GeeksforGeeks  
A computer science portal for geeks

122015115@sastru.ac.in

**GeeksforGeeks**  
A computer science portal for geeks

### Reverse Words in a String

```

void reverseWords(char str[], int n)
{
    int start = 0;
    for (int end = 0; end < n; end++)
    {
        if (str[end] == ' ')
        {
            reverse(str, start, end - 1);
            start = end + 1;
        }
    }
    reverse(str, start, n - 1);
    reverse(str, 0, n - 1);
}

void reverse(char str[], int low, int high)
{
    while (low <= high)
    {
        swap(str[low], str[high]);
        low++;
        high--;
    }
}

```

0 1 2 3 4 5 6 7 8 9 10 11 12 13  
 Welcome to Gfg  
 start = 0, n = 14

end = 0, 1, 2, 3, 4, 5, 6 : No change  
 end = 7 : emoclew to Gfg  
 start = 8  
 end = 8, 9 : No change  
 end = 10 : emoclew at Gfg  
 start = 11  
 end = 11, 12, 13 : No change

After Loop:  
 emoclew at gfh // Last word Reversed  
 Gfg to welcome // Whole String Reversed

Activate Windows  
[Go to Settings to activate Windows.](#)



## Pattern Searching

Overview:

122015115@sastru.ac.in

**GeeksforGeeks**  
A computer science portal for geeks

### Overview of Pattern Searching

I/p: txt = "GEEKSFORGEE KS"  
<sup>0 .. 2 .. . . . . 10 ..</sup>  
 pat = "EKS"

O/p: 2 10

I/p: txt = "AAAAA"  
<sup>0 1 2</sup>  
 pat = "AAA"

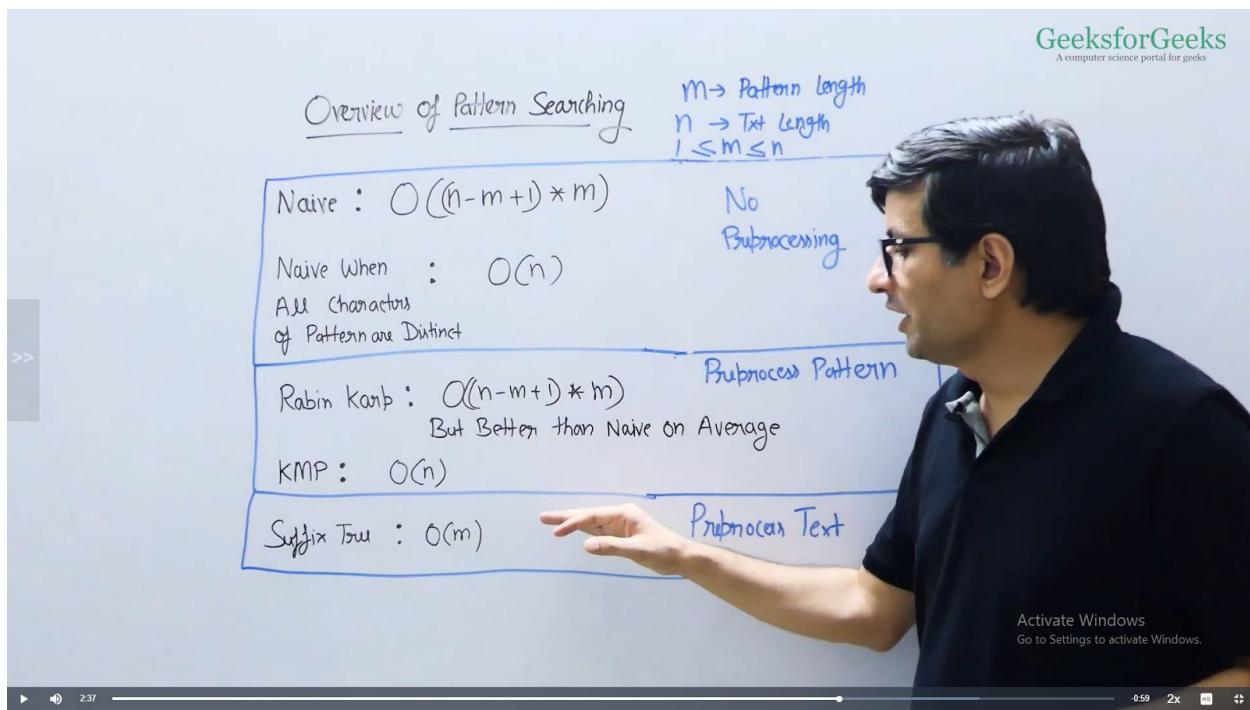
O/p: 0 1 2

I/p: txt = "ABCABCD"  
 pat = "ABD"  
 O/p: Not Present

Activate Windows  
[Go to Settings to activate Windows.](#)



Given two strings text and pattern, we need to find occurrences of pattern in the string and return their indices.



## Naive Pattern Searching

Given a pattern and a text, we need to print all occurrences of the pattern in the text. This takes about  $O((m+n-1)*m)$  time.

Naive Pattern Searching

I/p : txt = "ABABABCD"  
pat = "ABAB"

O/p : 0 2

>>

I/p : txt = "ABCABCD"  
pat = "ABCD"

O/p : 3

I/p : txt = "AAAAAA"  
pat = "AAA"

O/p : 0 1 2

Activate Windows  
Go to Settings to activate Windows.

3:49 2x

## Improved Pattern Matching with distinct elements in Pattern

Given a pattern with distinct characters and a text, we need to print all occurrences of the pattern in the text. Time: O(n)

Improved Naive Algorithm for Distinct  
↓  
in Pattern

I/p : txt = "ABCABCDABCD"  
pat = "ABCD"

O/p : 3 7

>>

I/p : txt = "GEEKSFORGEEKS"  
pat = "EKS"

O/p : 2 10

I/p : txt = "ABCAAAD"  
pat = "ABD"

O/p :

Activate Windows  
Go to Settings to activate Windows.

```

txt = 0 1 2 3 4 5 6 7 8
      ABC E A B E F A B C D
pat =          A B C D
n=12
m=4
i=0:   i= i+3
i=3:    O/p: 8
i=4: i= i+2
i=6: ...
i=7: ...
i=8

```

```

Void patSearchDist(txt, pat)
{
    int n = txt.length();
    int m = pat.length();
    for(int i = 0; i <= n-m; )
    {
        int j;
        for(j=0; j<m; j++)
            if(pat[j] != txt[i+j])
                break;
            if(j==m)
                print(i + " ");
            if(j==0)
                i++;
            else
                i = i + j;
    }
}

```

Activate Windows  
Go to Settings to activate Windows.

## Rabin Karp Algorithm

This also takes  $O((m + n - 1)*m)$  time similar to naive approach. But it is more efficient than naive on average.

Rabin Karp Algorithm

```

txt = "abdabcbabc"
pat = "abc"

```

- ① Like Naive algorithm, slide the pattern one by one.
- ② Compare hash values of pattern and current text window. If hash values match, then only compare individual characters.

Activate Windows  
Go to Settings to activate Windows.

- This algorithm works better than the naive algorithm for large patterns

- Unlike Naive algorithm, when we are at a current window in text, we do not compare all characters one by one immediately, rather we compare the hash value of the pattern with the current hash value of the text. If they match, then we compare individual characters and check if the pattern matches and print its index if it matches and move to next window

Rabin Karp Algorithm

txt = "a b d a b c b a b c"  
pat = "a b c"

Rolling Hash  
 $t_{i+1} = t_i + txt[i+m] - txt[i]$

M : length of Pattern

Simple Hash : Sum of values  
 Problem : Spurious Hits

a : 1	b : 2	c : 3	d : 4	e : 5
-------	-------	-------	-------	-------

GeeksforGeeks  
 A computer science portal for geeks

$\sum = (1+2+3) = 6$

$i = 0 : t = (1+2+4) = 7$   
 $i = 1 : t = (2+4+1) = 7$   
 $i = 2 : t = (4+1+2) = 7$   
 $i = 3 : t = (1+2+3) = 6 \text{ (Match)}$   
 $i = 4 : t = (2+3+2) = 7$   
 $i = 5 : t = (3+2+1) = 6 \text{ (Spurious Hit)}$   
 $i = 6 : t = (2+1+2) = 5$   
 $i = 7 : t = (1+2+3) = 6 \text{ (Match)}$

Activate Windows  
 Go to Settings to activate Windows.

3:58 - 7:15 2x

- The problem with above approach is spurious hits. Those occur when two hash values match but the pattern doesn't match with the text window. For example, the hash value for abc(1 + 2 + 3) matches with cba(3 + 2 + 1).
- So we change the hash function and introduce weighted sum
- For simplicity, we are taking  $d = 5$  as we consider set of 5 characters from which the input is drawn

Improved Hash

$$h("abc") = 1 \times d^2 + 2 \times d^1 + 3 \times d^0 = 1 \times 5^2 + 2 \times 5^1 + 3 \times 5^0 = 38$$

$$d=5$$

$$h("dab") = 4 \times d^2 + 1 \times d^1 + 2 \times d^0 = 4 \times 5^2 + 1 \times 5^1 + 2 \times 5^0 = 107$$

Rolling Hash:

$$t_{i+1} = d(t_i - txt[i] \times d^{M-1}) + txt[i+M]$$

$M \rightarrow$  Length of pattern

Example:

$$txt = "a b c d a b c b a b c"$$

$$t_0 = 1 \times 5^2 + 2 \times 5^1 + 3 \times 5^0 = 39$$

$$t_1 = 5 \times (t_0 - 1 \times 5^2) + 1 = 71$$

$$t_2 = 5 \times (t_1 - 2 \times 5^2) + 2 = 107$$

$$t_3 = 5 \times (t_2 - 3 \times 5^2) + 3 = 38$$

$$t_4 = 5 \times (t_3 - 4 \times 5^2) + 4 = 107$$

$$t_5 = 5 \times (t_4 - 5 \times 5^2) + 5 = 38$$

$$t_6 = 5 \times (t_5 - 6 \times 5^2) + 6 = 107$$

$$t_7 = 5 \times (t_6 - 7 \times 5^2) + 7 = 38$$

$$t_8 = 5 \times (t_7 - 8 \times 5^2) + 8 = 107$$

$$t_9 = 5 \times (t_8 - 9 \times 5^2) + 9 = 38$$

bat = "abc"  
 $b = 1 \times 5^2 + 2 \times 5^1 + 3 \times 5^0 = 38$

Activate Windows  
Go to Settings to activate Windows.

Void RBSearch(bat, txt, M, N)

```

int h=1;
for(int i=1; i<=M-1; i++)
    h = (h * d) % q;

int p=0, t=0;
for(int i=0; i<M; i++)
{
    p = (p * d + bat[i]) % q;
    t = (t * d + txt[i]) % q;
}

for(int i=0; i<=N-M; i++)
{
    if(p==t)
    {
        bool flag = true;
        for(int j=0; j<M; j++)
            if(txt[i+j] != bat[j]) {flag = false; break;}
        if(flag == true) {cout << i; }
    }
}

```

Compute  $(d^{M-1}) \times q$

Compute  $p$  and  $t_0$  →

Check for Spurious hit →

Compute  $t_{i+1}$  using  $t_i$  →

Activate Windows  
Go to Settings to activate Windows.

- The reason for doing computations under modulo q is that the computation of weightsum of ASCII values, the sum may go large and so we do modulo to store it in an int or long long int variable. Generally 'q' is a large prime number and it is taken as large as possible to reduce spurious hits

## Construction of LPS Array:

LPS => Longest Proper Prefix Suffix Array

- Proper prefixes are those substrings which are of length ranging from 1 to one less than length of the string.
- Suffixes are those substrings which are of length ranging 1 to length of the string.
- In LPS array, we take the length of the Longest Proper Prefix Suffix string

Constructing Longest Proper Prefix Suffix Array

I/p: str = "ababc"  
O/p: lps[] = {0, 0, 1, 2, 0}

I/p: str = "aaaaa"  
O/p: lps[] = {0, 1, 2, 3}

I/p: str = "abcd"  
O/p: lps[] = {0, 0, 0, 0}

I/p: str = "ababab"  
O/p: lps[] = {0, 0, 1, 2, 0}

Proper Prefixes of "abcd"  
", "a", "ab", "abc"  
Suffixes of "abcd"  
" ", "d", "cd", "bcd", "abcd"

Activate Windows  
Go to Settings to activate Windows.



Constructing Longest Proper Prefix Suffix Array

I/p: str = "ababc"  
O/p: lps[] = {0, 0, 1, 2, 0}

I/p: str = "aaaaa"  
O/p: lps[] = {0, 1, 2, 3}

I/p: str = "abcd"  
O/p: lps[] = {0, 0, 0, 0}

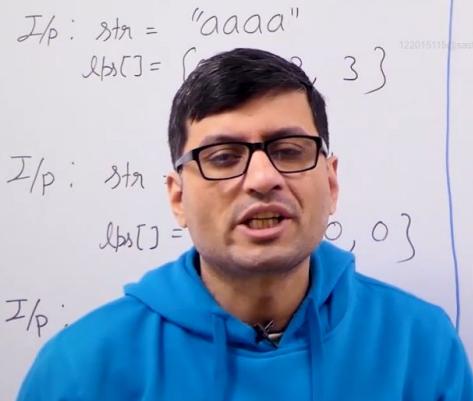
I/p: str = "ababab"  
O/p: lps[] = {0, 0, 1, 2, 0}

Exercise  
str = "abacabad"  
lps[] = {0, 0, 1, 0, 1, 2, 3, 0}

str = "abbabb"  
lps[] = {0, 0, 0, 1, 2, 3}

I/p: str = "abacabad"  
O/p: lps[] = {0, 0, 1, 0, 1, 2, 3, 0}

Activate Windows  
Go to Settings to activate Windows.



## Naive Solution: O(n^3)

Naive : O(n<sup>3</sup>)

str = ababacab  
len = 7      N = 8

str[0], str[1]  
str[1], str[2]  
⋮  
str[6],  
len = 6  
str[0],  
str[1],  
⋮  
str[5],  
len  
⋮  
str[7]



int longProbPreSuff(str, n)  
{  
 for(int len = n-1; len > 0; len--){  
 bool flag = true;  
 for(int i=0; i < len; i++)  
 if(str[i] != str[n-len+i])  
 flag = false;  
 if(flag == true)  
 return len;  
 }  
 return 0;  
}  
void fillLPS(str, lps[]){  
 for(int i=0; i < str.length(); i++)  
 lps[i] = longProbPreSuff(str, i+1);  
}

Activate Windows  
Go to Settings to activate Windows.

## Optimized Solution: O(n)

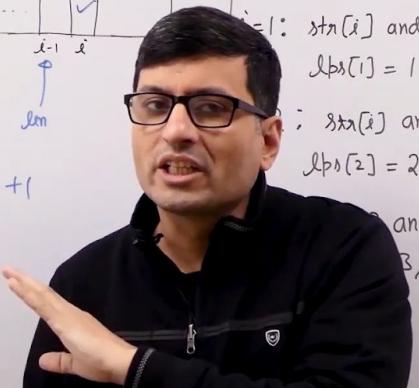
Efficient : O(n)

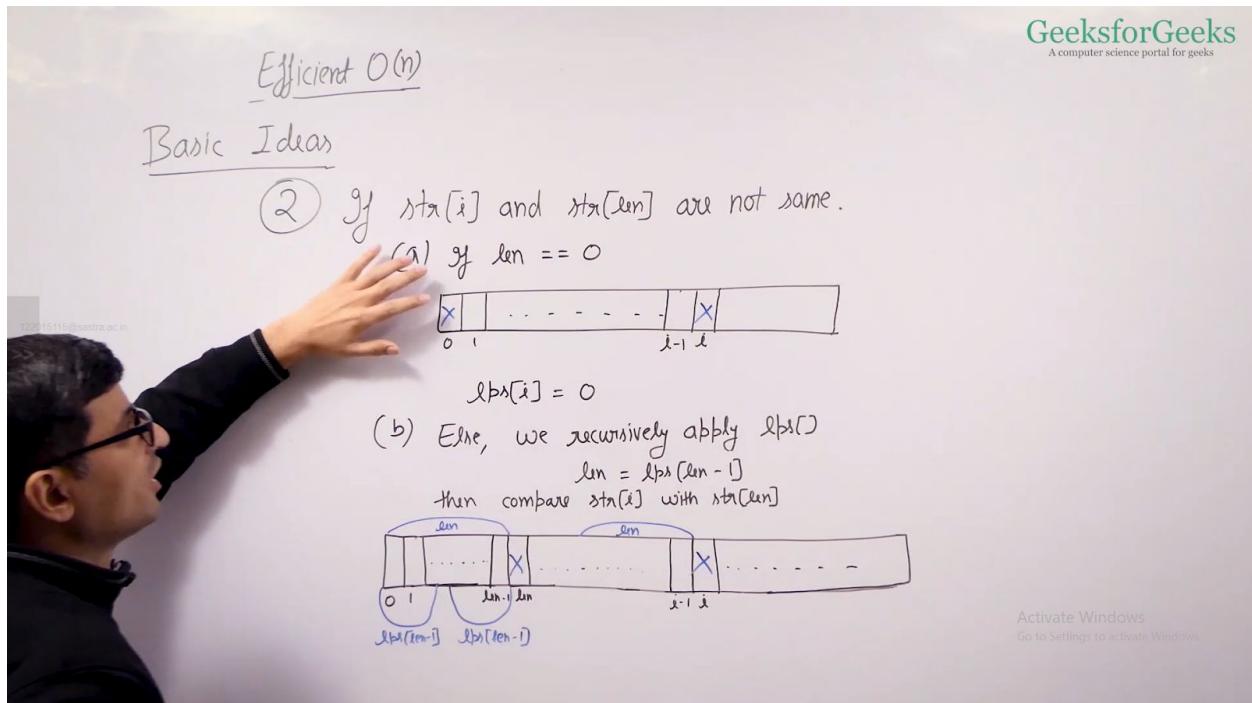
Basic Idea :

① If len = lps[i-1] and str[len] and str[i] are same, then lps[i] = len + 1

str = a a a a  
lps = 0 1 2 3  
i=0: len = 0, lps[0]  
i=1: str[i] and str[len] are same  
lps[1] = 1, len = 1  
i=2: str[i] and str[len] are same  
lps[2] = 2, len = 2  
i=3: str[i] and str[len] are same  
lps[3] = 3, len = 3

Activate Windows  
Go to Settings to activate Windows.





Activate Windows  
Go to Settings to activate Windows.

Implementation of O(n) algorithm:

String: 0 1 2 3 4 5 6 7 8  
 $str = AAAABAAC$

$len = 0, lps[0] = 0$

$i=1: lps[1] = 1, len = 1$

$i=2: lps[2] = 2, len = 2$

$i=3: lps[3] = 2$   
 $str[i] \neq str[len]$  and  $len > 0$   
 $len = lps[2] = 2$   
 again  $str[i] = A$   
 $len = 1$   
 $lps[3] = 1$

$i=4: lps[4] = 0$

$i=5: lps[5] = 1$

$i=6: lps[6] = 0$

$i=7: lps[7] = 0$

$i=8: lps[8] = 0$

Code:

```
Void fillLPS(str, lps[])
{
    int n = str.length(), len=0;
    lps[0] = 0;
    int i=1;
    while(i < n)
    {
        if(str[i] == str[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if(len == 0) { lps[i] = 0; i++; }
            else { len = lps[len-1]; }
        }
    }
}
```

$i=8: lps[8]$   
 $len = lps[2] = 2$   
 $len = lps[1] = 1$   
 $len = lps[0] = 0$   
 $lps[8] = 0$

Activate Windows  
Go to Settings to activate Windows.

## KMP Algorithm

- For using KMP algorithm, we should know to construct LPS(Longest properPrefix Suffix) array

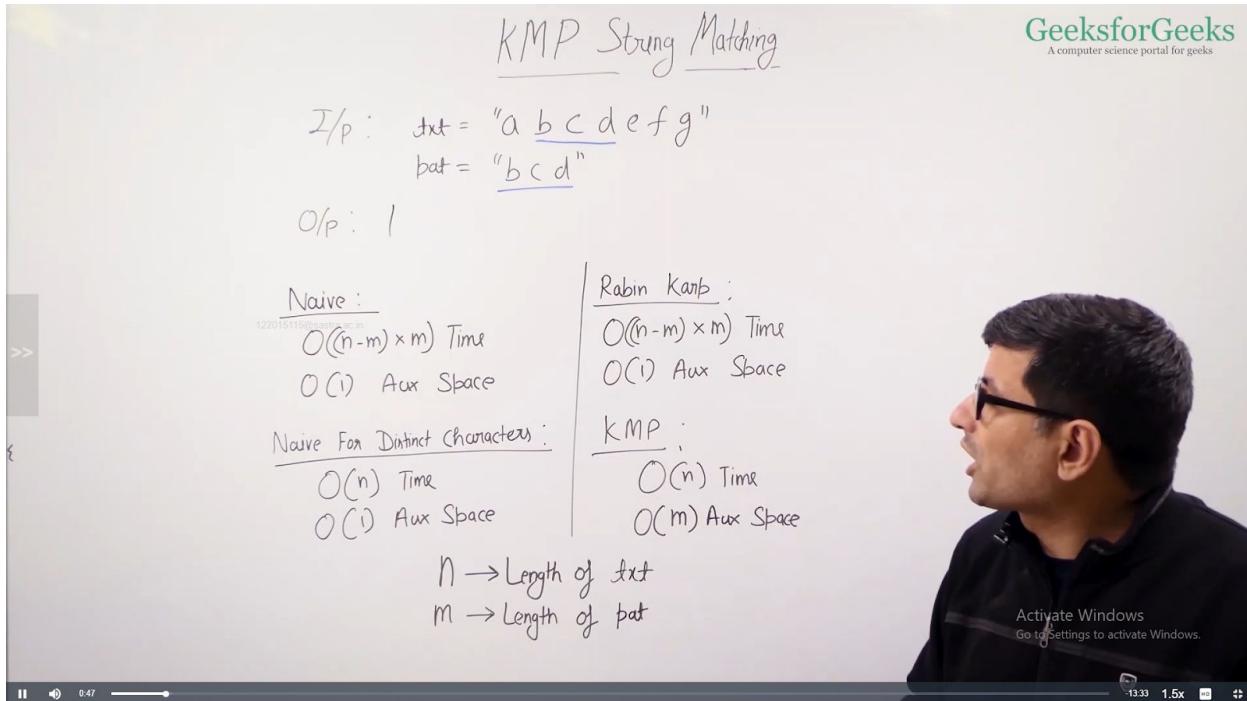
KMP String Matching

I/p : txt = "a b c d e f g"  
pat = "b c d"

O/p : |

<u>Naive :</u> $O((n-m) \times m)$ Time $O(1)$ Aux Space	<u>Rabin Karp :</u> $O((n-m) \times m)$ Time $O(1)$ Aux Space
<u>Naive For Distinct Characters :</u> $O(n)$ Time $O(1)$ Aux Space	<u>KMP :</u> $O(n)$ Time $O(m)$ Aux Space

$n \rightarrow$  Length of txt  
 $m \rightarrow$  Length of pat



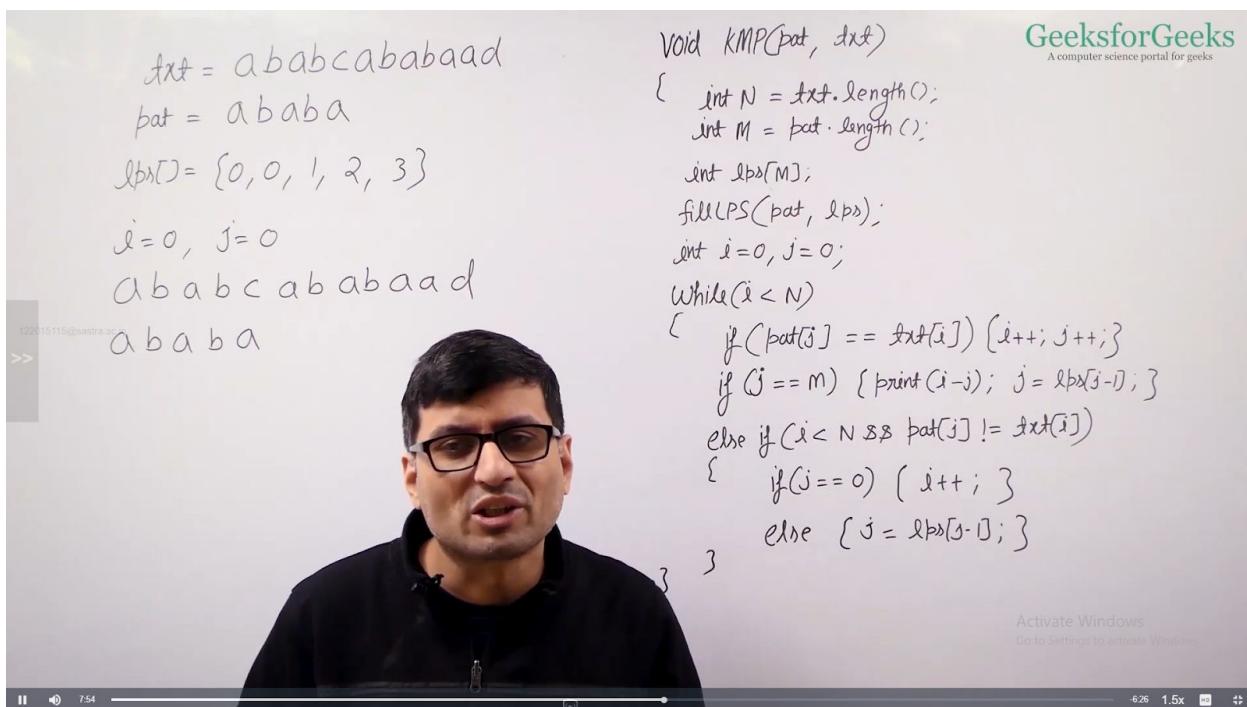
Activate Windows  
Go to Settings to activate Windows.

txt = ababcababaad  
pat = ababa  
lps[] = {0, 0, 1, 2, 3}

$i=0, j=0$

ababc ababaad  
ababa

void KMP(pat, txt)  
{  
 int N = txt.length();  
 int M = pat.length();  
 int lps[M];  
 fillLPS(pat, lps);  
 int i=0, j=0;  
 while(i < N)  
 {  
 if (pat[j] == txt[i]) {i++; j++;}  
 if (j == M) {print(i-j); j = lps[j-1];}  
 else if (i < N && pat[j] != txt[i])  
 {  
 if (j == 0) {i++};  
 else {j = lps[j-1];}  
 }  
 }  
}



Activate Windows  
Go to Settings to activate Windows.

122015115@sastrac.ac.in

a b a b c a b a b a a d

a b a b a       $i=0, j=0$        $i=4, j=4$

a b a b c a b a b a a d

a b a b a       $i=4, j=2$

a b a b c a b a b a a d

a b a b a       $i=4, j=0$

a b a b c a b a b a a d

a b a b a       $i=5$        $j=0$

a b a b c a b a b a a d

a b a b a       $i=10$        $j=3$

a b a b c a b a b a a d

a b a b a       $i=10, j=1$

a b a b c a b a b a a d

a b a b a       $i=10, j=0$

a b a b c a b a b a a d

a b a b a       $i=11, j=0$

GeeksforGeeks  
A computer science portal for geeks

```
Void KMP(pat, txt)
{
    int N = txt.length();
    int M = pat.length();
    int lps[M];
    fillLPS(pat, lps);
    int i=0, j=0;
    while(i < N)
    {
        if(pat[i] == txt[j]) {i++; j++;}
        if(j == M) {i=j-lps[M-1]; j=0;}
        else if(pat[i] < txt[j])
        {
            if(j != 0) j = lps[j-1];
            else i++;
        }
        else {i++; j++;}
    }
}
```

Activate Windows  
Go to Settings to activate

13:05 13:32  
1:15 1.5x

## Check if Strings are Rotations

122015115@sastrac.ac.in

Check for Rotation

I/p :  $s_1 = "ABCD"$ ,  $s_2 = "CDAB"$

O/p : Yes       $\text{ABCD} \rightarrow \text{BCDA} \rightarrow \text{CDAB}$

I/p :  $s_1 = "ABAAB"$ ,  $s_2 = "BAAAA"$

O/p : Yes       $\text{ABAAB} \rightarrow \text{BAAAA}$

I/p :  $s_1 = "ABAB"$ ,  $s_2 = "ABBA"$

O/p : No

GeeksforGeeks  
A computer science portal for geeks

Activate Windows  
Go to Settings to activate Windows.

Naive Approach:  $O(n^2)$

```
string lrotate(string& str, int k)
{
    string result;
    for(int i = k; i < str.length(); i++)
        result += str[i];
    for(int i = 0; i < k; i++)
        result += str[i];
    return result;
}
bool checkRotation(string& str1, string& str2)
{
    //Naive approach: O(n^2)
    for(int i = 0; i < str1.size(); i++)
    {
        if(str2 == lrotate(str1, i))
            return true;
    }
    return false;
}
```

Efficient approach:  $O(n)$

The image shows a man in a black polo shirt standing in front of a whiteboard, gesturing with his hands as if explaining something. He is positioned to the right of a whiteboard that displays two pieces of code and some handwritten notes.

**Code Snippet 1 (C++):**

```
bool areRotations(string s1, string s2)
{
    if(s1.length() != s2.length()) return false; C++
    return ((s1+s1).find(s2) != string::npos);
}
```

**Code Snippet 2 (Java):**

```
>>> boolean areRotations(String s1, String s2)
{
    if(s1.length() != s2.length()) return false; Java
    return ((s1+s1).indexOf(s2) >= 0);
}
```

**Handwritten Notes on the Whiteboard:**

- $s_1$
- $s_2$
- $s_1 + s_1$
- $s_1 = ABCD$
- $s_2 = BCDA$
- $s_1 + s_1 = ABCDABCD$

**GeeksforGeeks Logo:**

GeeksforGeeks  
A computer science portal for geeks

**Activation Message:**

Activate Windows  
Go to Settings to activate Windows.

## Anagram Search

Given a text and a pattern, the task is to find if there is an anagram of pattern present in text.

The video thumbnail shows a person in a black shirt and glasses, holding a blue marker, standing in front of a whiteboard. On the whiteboard, there is handwritten text:

Anagram Search

I/p : txt = "geeksforgeeks"  
pat = "frog"  
O/p : Yes

I/p : txt = "geeksforgeeks"  
pat = "neek"

O/p : No

At the top right of the whiteboard, it says "GeeksforGeeks A computer science portal for geeks". At the bottom right, there is a watermark for "Activate Windows Go to Settings to activate Windows." A progress bar at the bottom indicates the video is at 0:56.

The video thumbnail shows a person in a black shirt and glasses, pointing with their right hand. On the whiteboard, there is handwritten text:

Naive Solution

C++ code:

```
bool isPresent(string &txt, string &pat)
{
    int n = txt.length();
    int m = pat.length();
    for (int i=0; i<=n-m; i++)
    {
        if (isAnagram(pat, txt, i))
            return true;
    }
    return false;
}

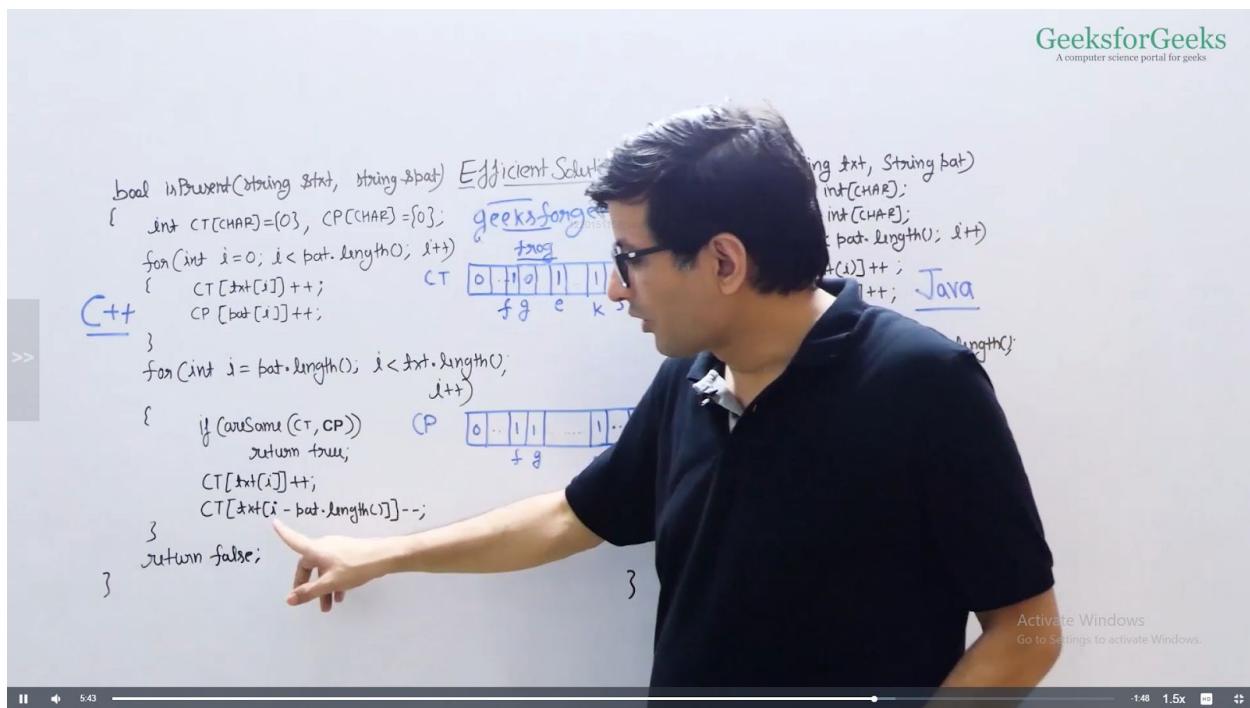
bool isAnagram(string &pat, string &txt,
               int i)
{
    int count[CHAR] = {0};
    for (int j=0; j<txt.length(); j++)
    {
        count[pat[i]]++;
        count[txt[i+j]]--;
    }
    for (int j=0; j<CHAR; j++)
        if (count[j] != 0)
            return false;
    return true;
}
```

Java code:

```
boolean isPresent(String txt, String pat)
{
    int n = txt.length();
    int m = pat.length();
    for (int i=0; i<=n-m; i++)
    {
        if (isAnagram(pat, txt, i))
            return true;
    }
    return false;
}

boolean isAnagram(String pat, String txt,
                  int i)
{
    int count[] = new int[CHAR];
    for (int j=0; j<pat.length(); j++)
    {
        count[pat[i]]++;
        count[txt[i+j]]--;
    }
    for (int j=0; j<CHAR; j++)
        if (count[j] != 0)
            return false;
    return true;
}
```

At the top right of the whiteboard, it says "GeeksforGeeks A computer science portal for geeks". At the bottom right, there is a watermark for "Activate Windows Go to Settings to activate Windows."



## Smallest window in a string containing all the characters of another string

Given two strings. Find the smallest window in the first string consisting of all the characters of the second string.

Req Complexity :  $O(|S|)$  time and  $O(1)$  space

### **Input:**

S = timetopractice

P = toc

### **Output:** toprac

**Explanation:** toprac is the smallest subset in which toc can be found.

### **Input:**

S = zoomlazapzo

P = oza

### **Output:** apzo

**Explanation:** To find oza in the zoomlazapzo the smallest subset is

apzo.

```
9 // return the smallest window in s with all the characters of p
10 // if no such window exists, return "-1"
11 string smallestWindow (string s, string p){
12     // Your code here
13
14     int m = p.length();
15     int n = s.length();
16     if(m > n)
17         return "-1";
18     int hash_pat[256] = {0};
19     int hash_txt[256] = {0};
20     for(int i = 0; i < m; i++)
21     {
22         hash_pat[p[i]]++;
23     }
24     int start = 0;
25     int start_index = -1;
26     int counter = 0;
27     int min_length = INT_MAX;
28     for(int i = 0; i < n; i++)
29     {
30         hash_txt[s[i]]++;
31         if(hash_txt[s[i]] <= hash_pat[s[i]])
32         {
33             counter++;
34         }
35         if(counter == m)
36         {
37             while(hash_txt[s[start]] > hash_pat[s[start]] || hash_pat[s[start]] == 0)
38             {
39                 if(hash_txt[s[start]] > hash_pat[s[start]])
40                     hash_txt[s[start]]--;
41                 start++;
42             }
43             int len_window = i - start + 1;
44             if(min_length > len_window)
45             {
46                 min_length = len_window;
47                 start_index = start;
48             }
49         }
50     }
51     if(start_index == -1)
52         return "-1";
53     return s.substr(start_index, min_length);
54 }
55 } // Driver Code Ends
```

Activate Windows  
Go to Settings to activate Windows.