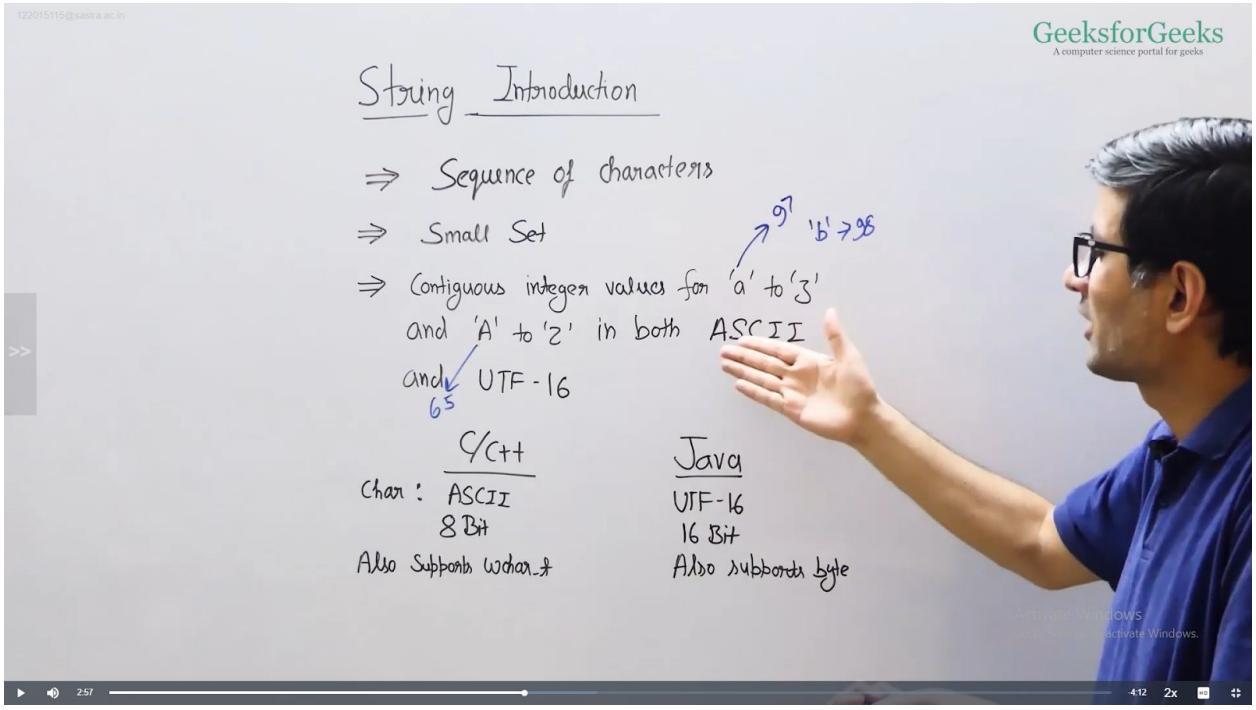
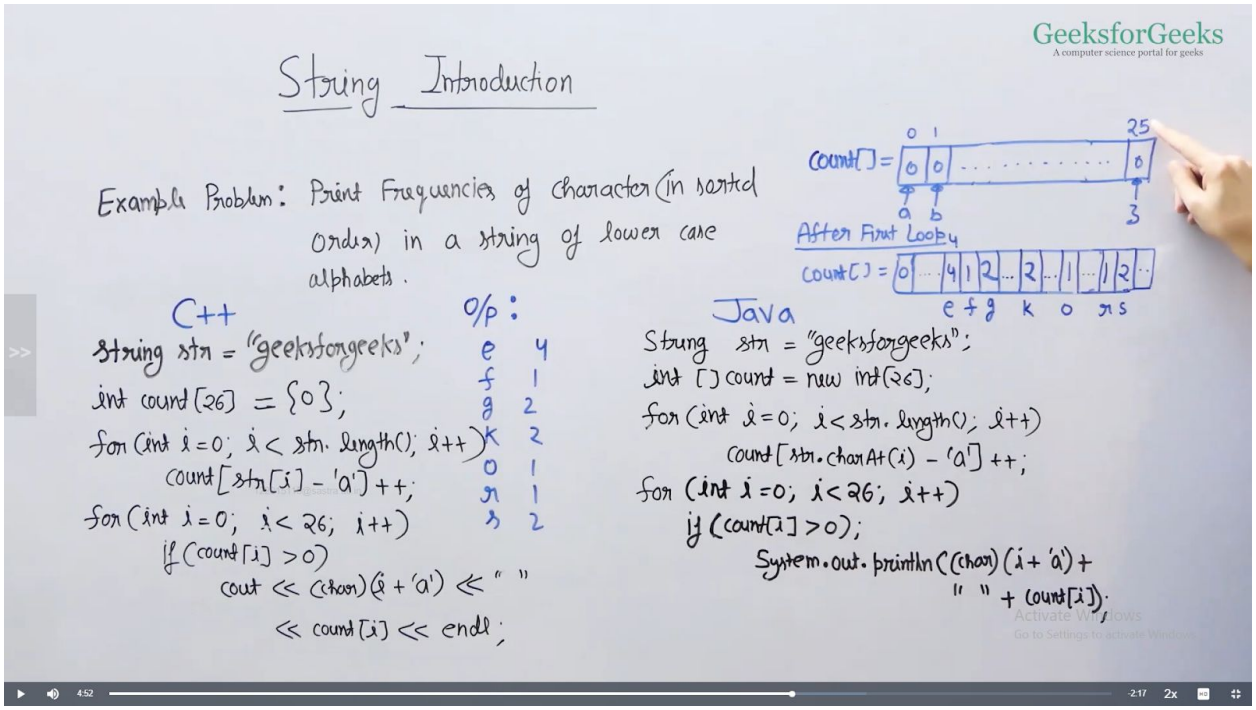
Strings

Introduction

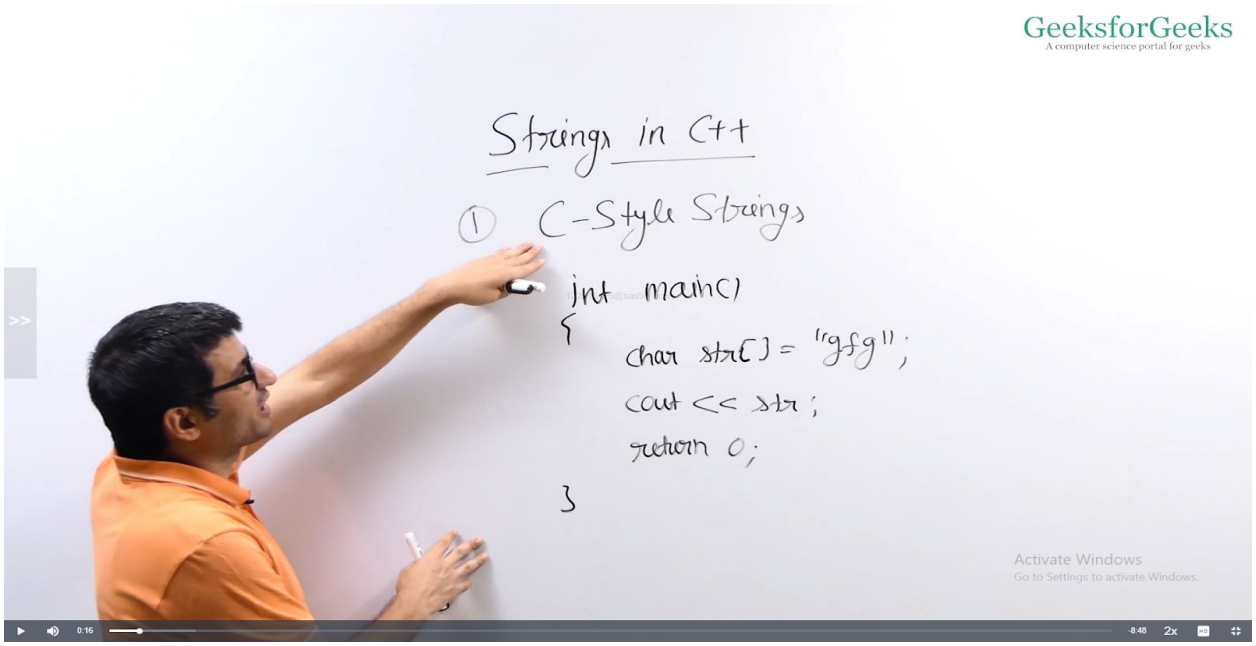


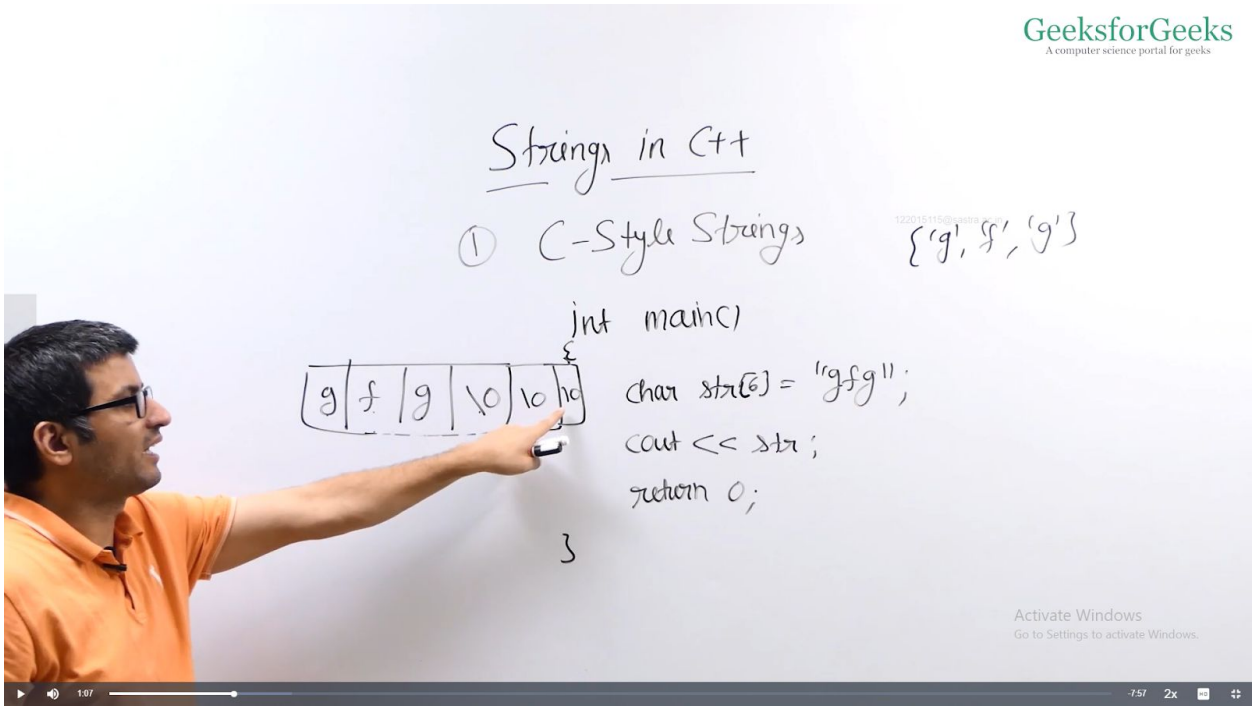
- Strings are collections of characters…. For C++, 8Bit ASCII and for Java, 16Bit UTF-16 code. Both ASCII and UTF-16 have the same encoding for the first 128 characters. For storing additional languages UTF-16 has another 8 bits extra



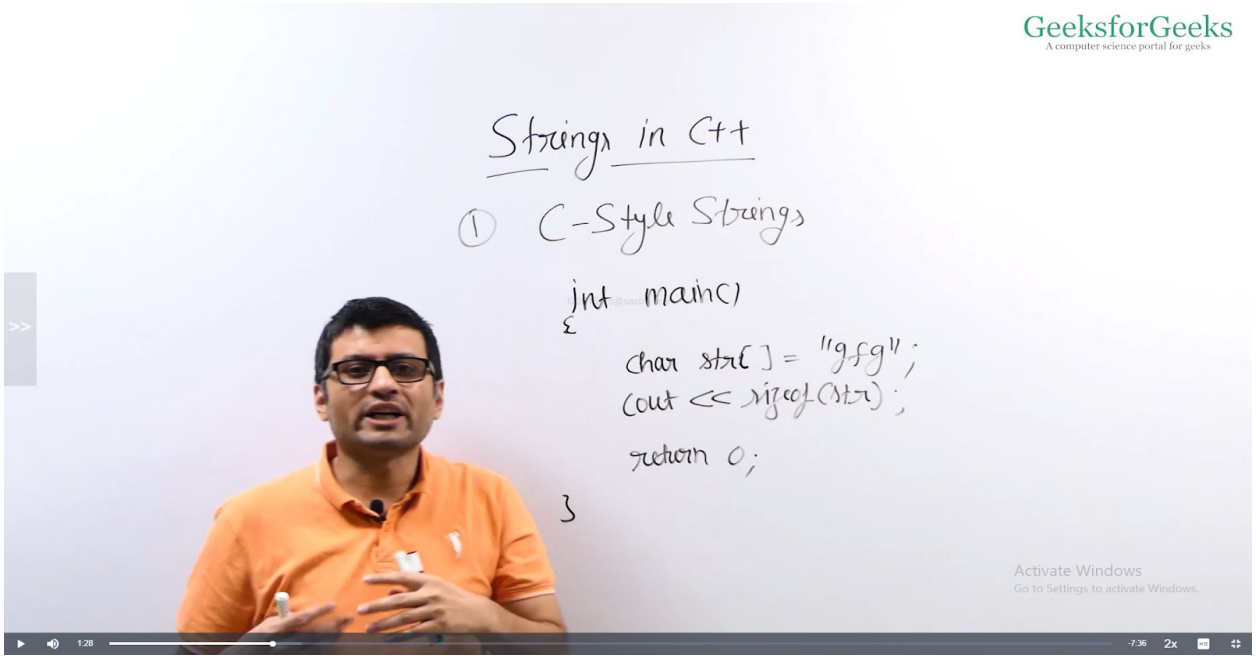
Strings in C++

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

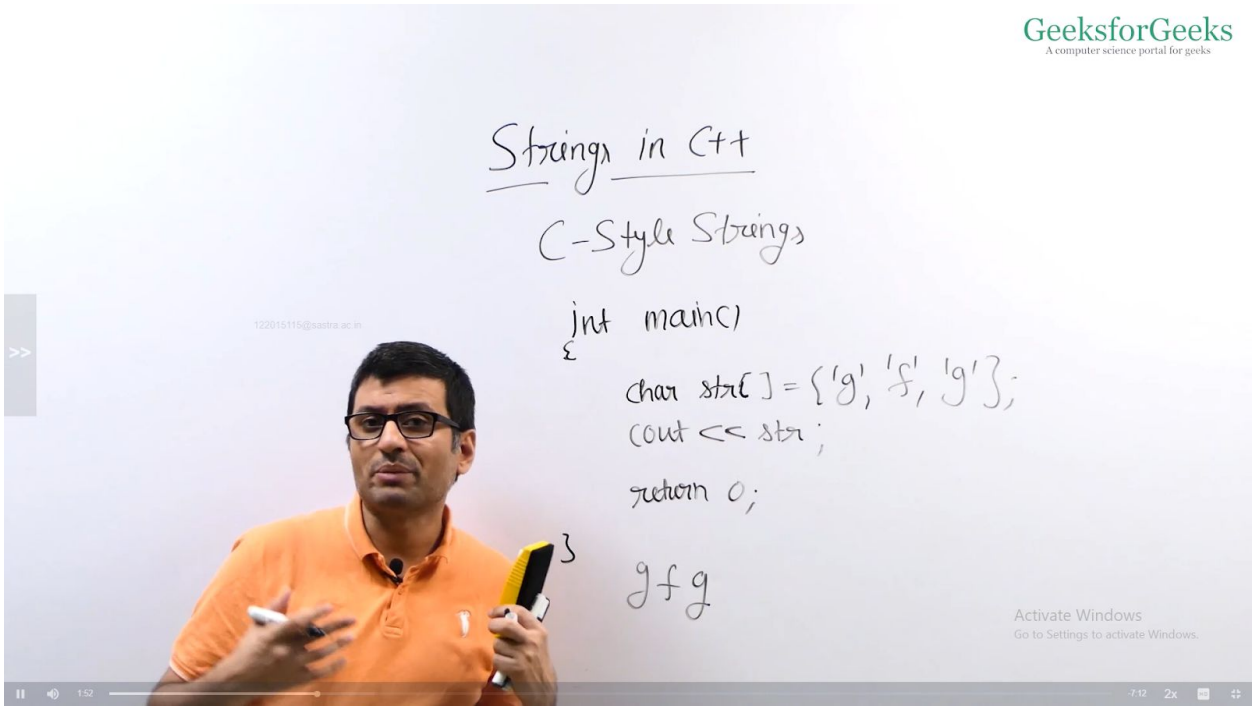




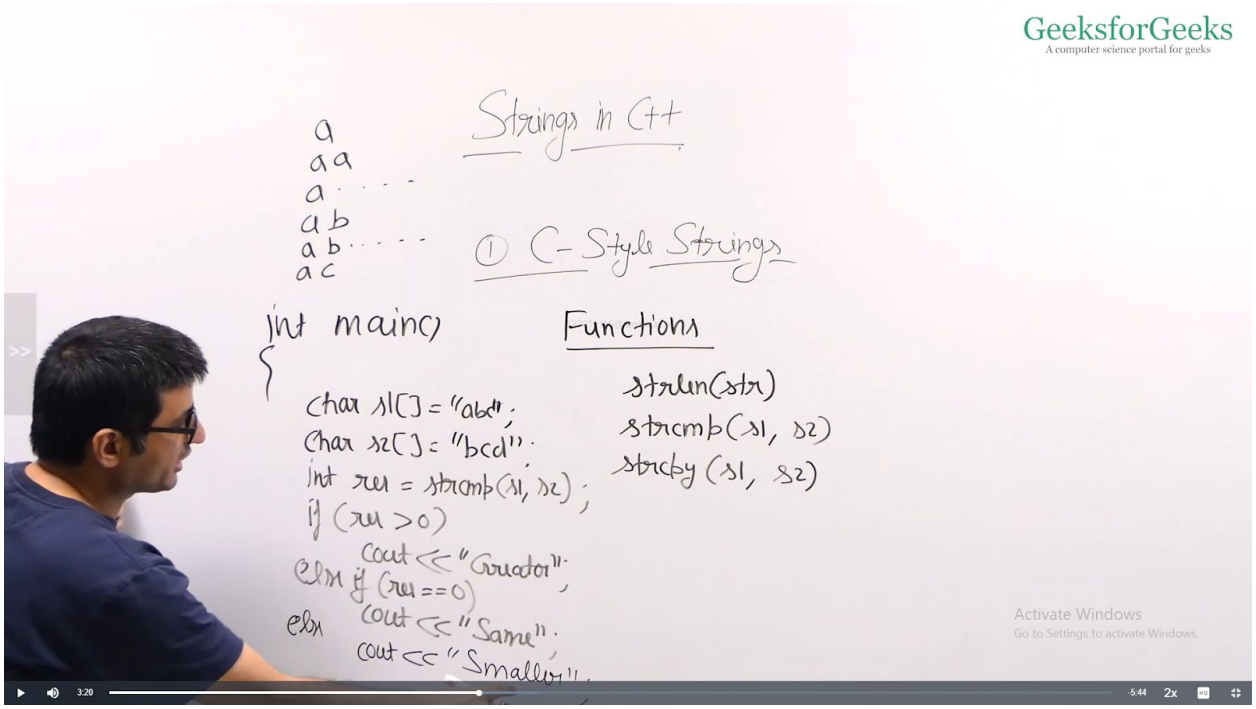
- If we initialize with more size, then all other cells will be filled with '\0' char.



- 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.

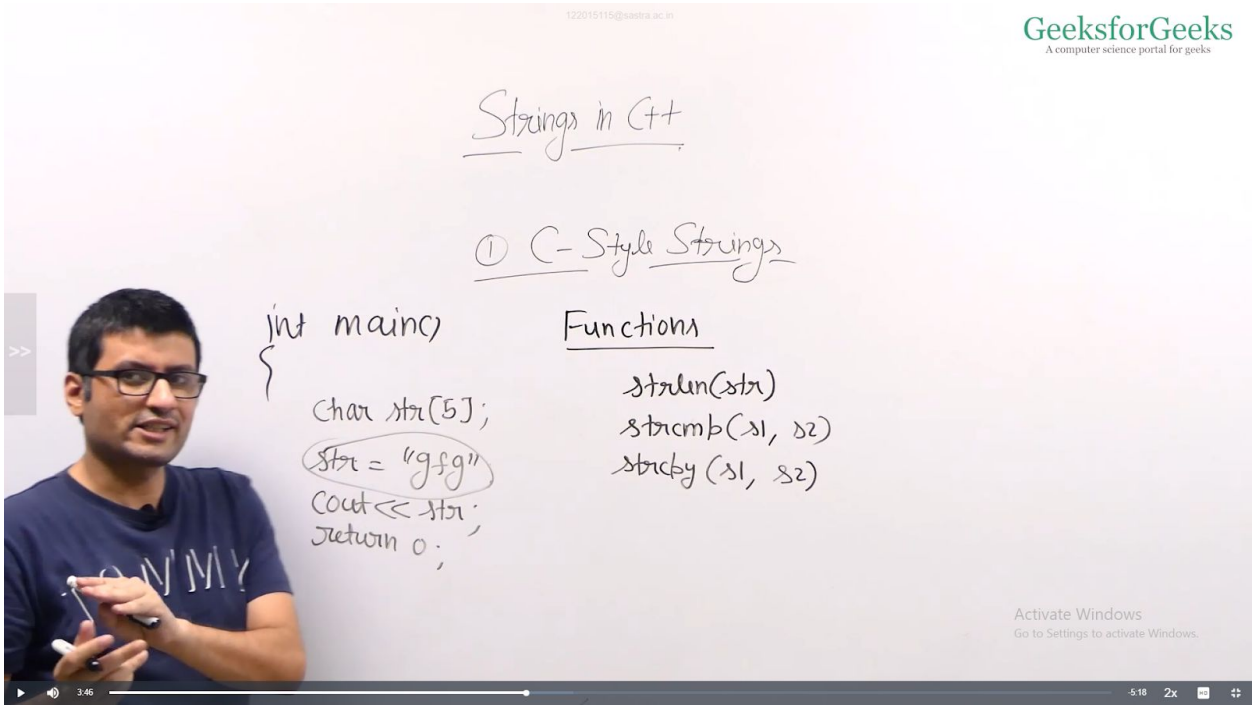


- Prints gfg along with some random characters or segmentation fault or gfg alone becz when initializing as a char array we need to put '\0' explicitly. If we initialize as char str[] = "gfg", the compiler automatically puts '\0' at the end

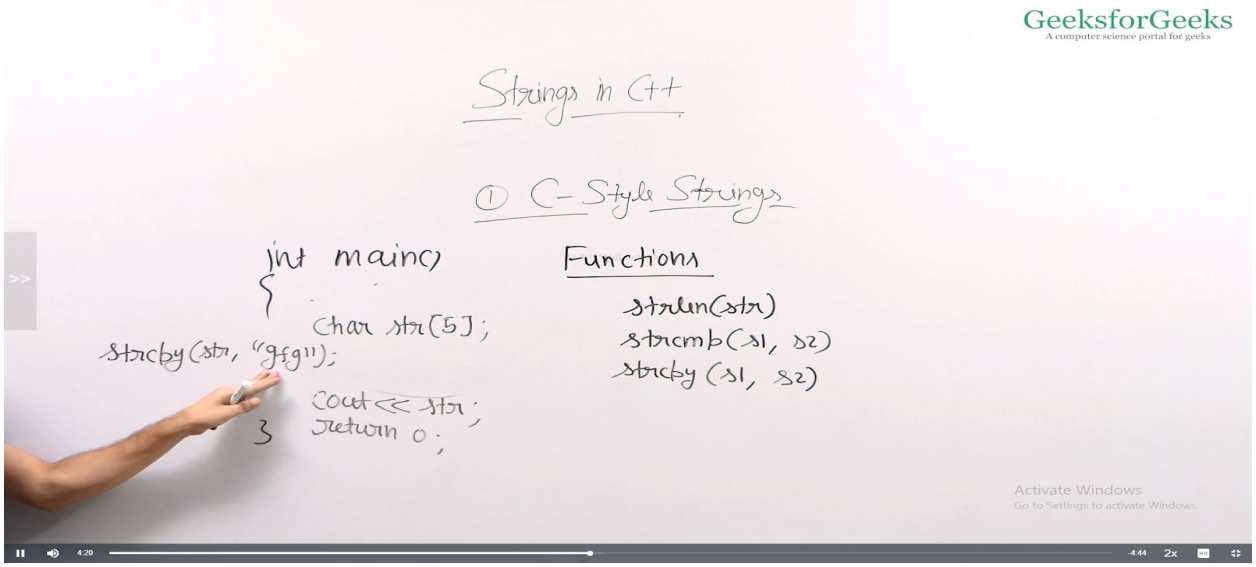


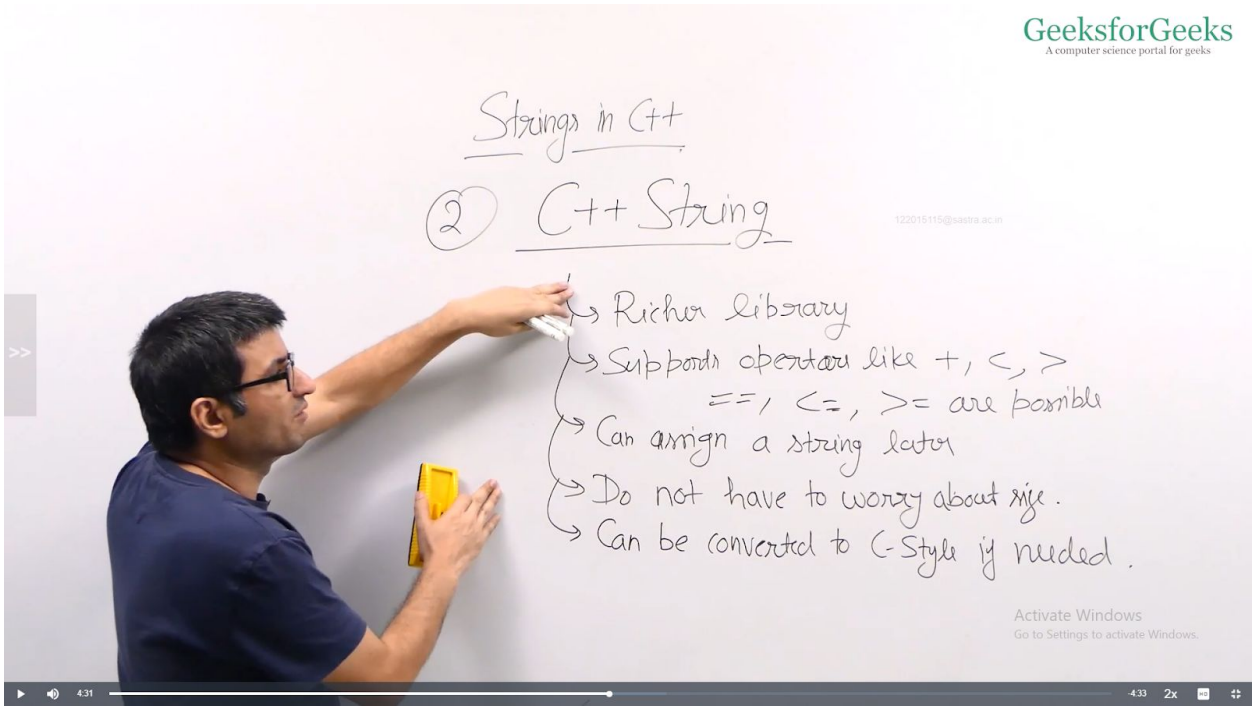
- strlen(str) gives length of the string excluding '\0' char.

- 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.

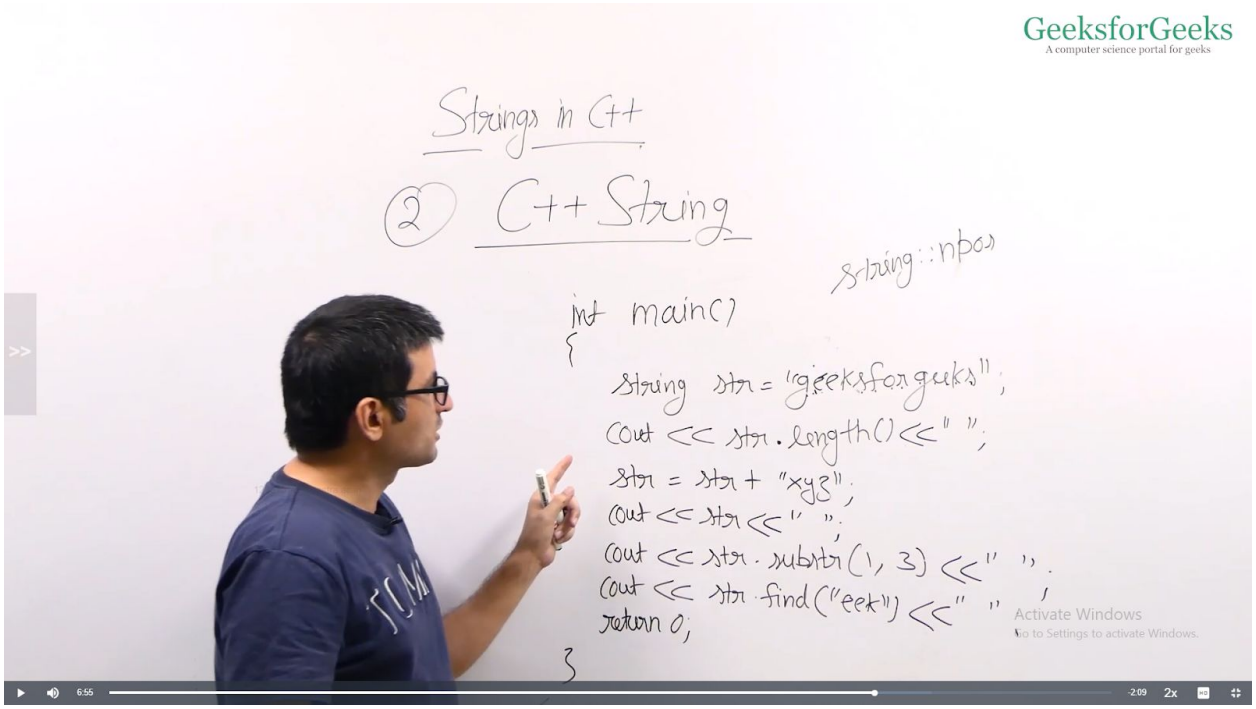


- The program will generate compiler error because str is a char array, str gives the address of the array and thus it can't be used in the left hand side of the assignment statement. So here we need strcpy()



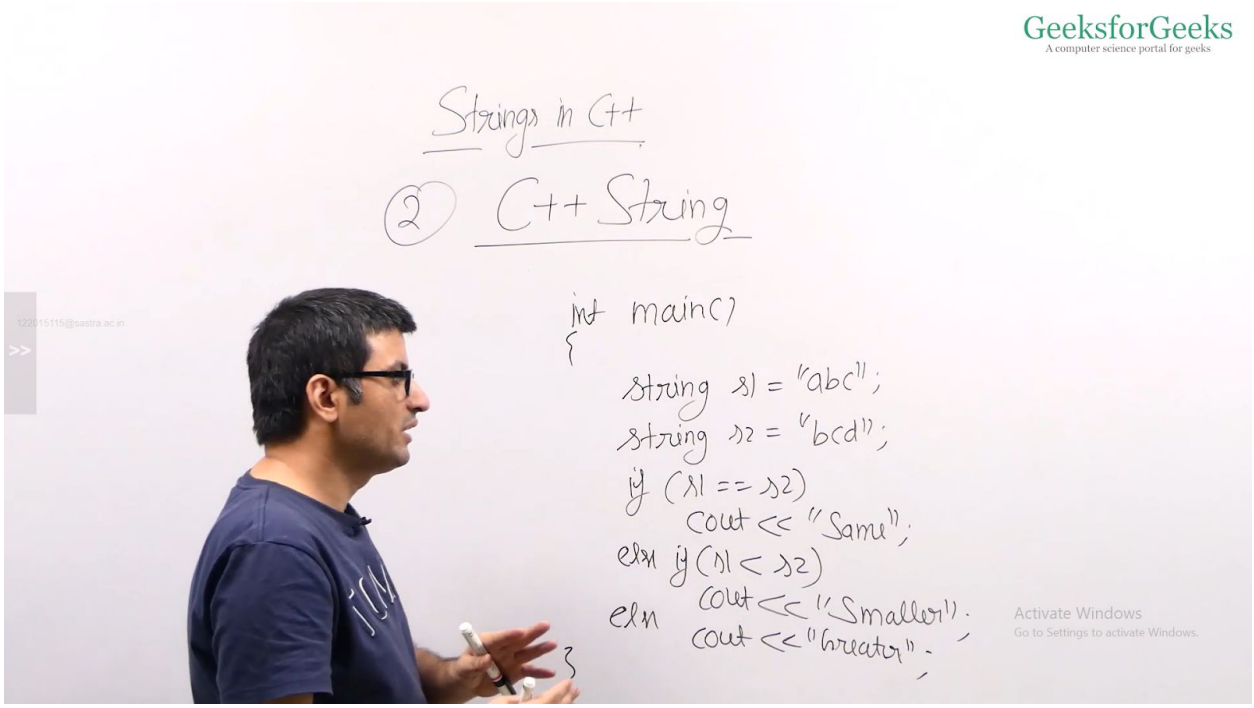


- Easier to use C++ style strings
- String is in-built class in C++

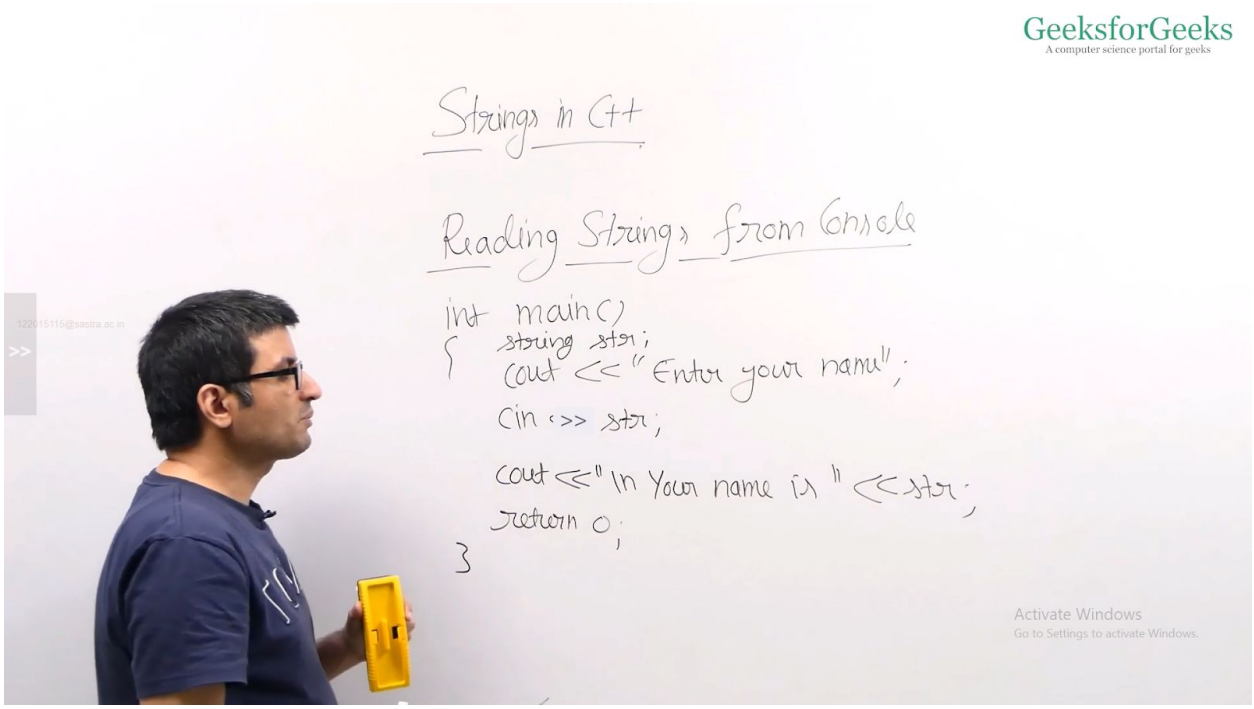


Output: 13 geeksforgeeksxyz eek 1

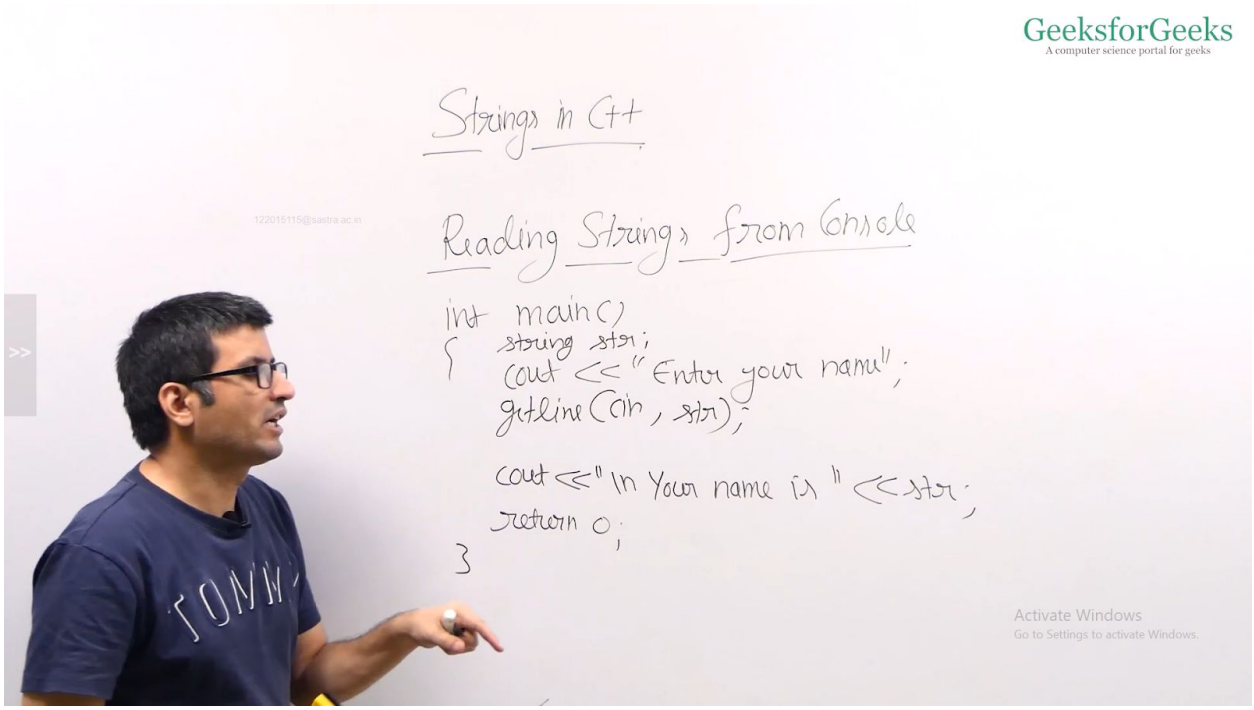
- str.find(str1) gives index of str1 in str. If not present if will return string::npos
- str.find(str1, pos) gives index of str1 in str after pos if it is present(same as java)



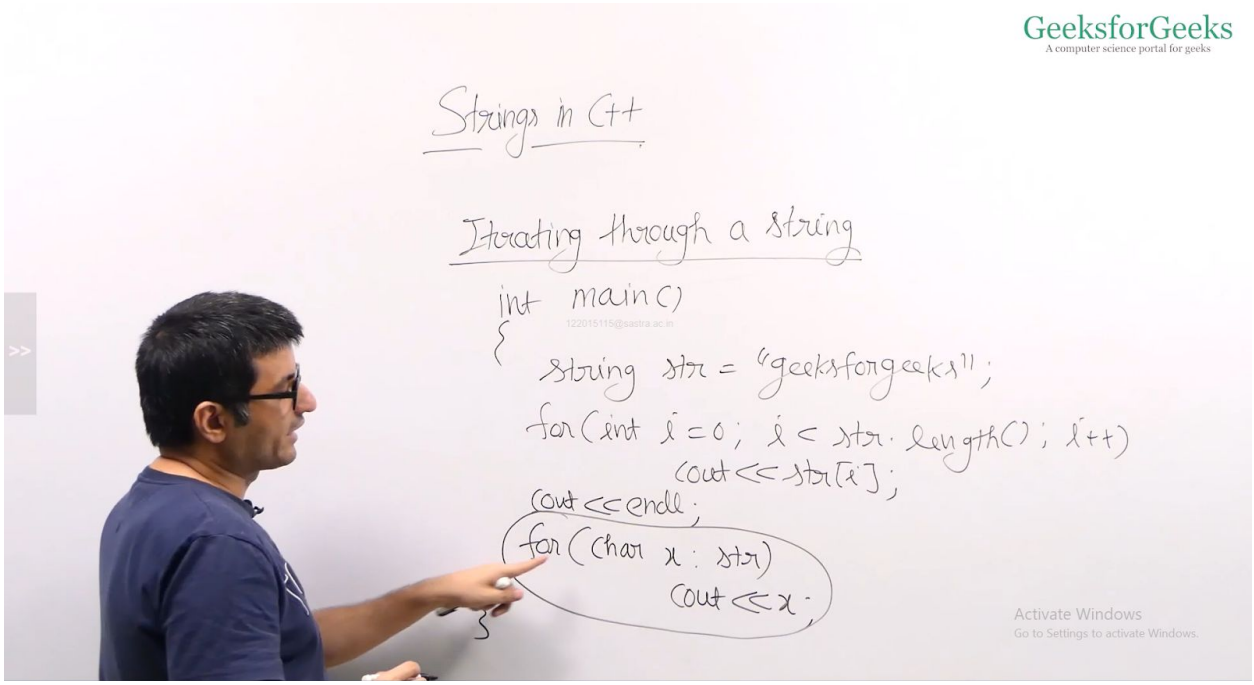
- Usage of comparison operators



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



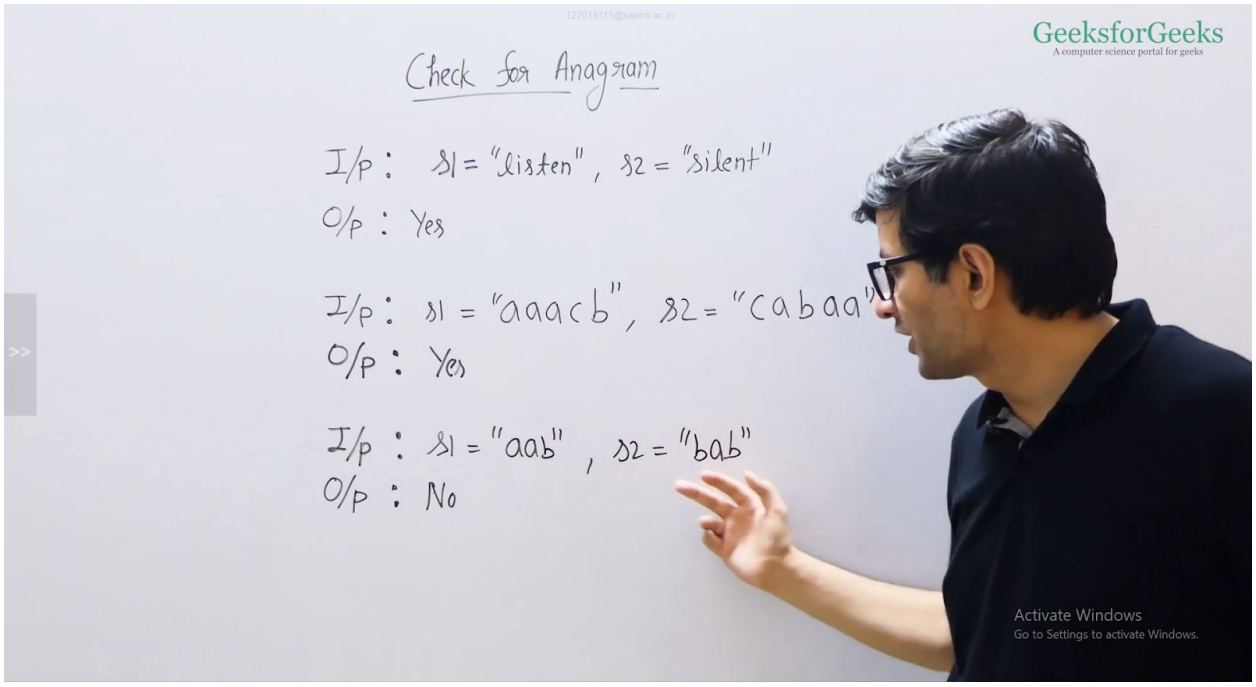
- This will take input till enter is pressed
- getline(cin, str, '$') will stop taking input if we press '$' only. By default, the 3rd parameter will be '\n'

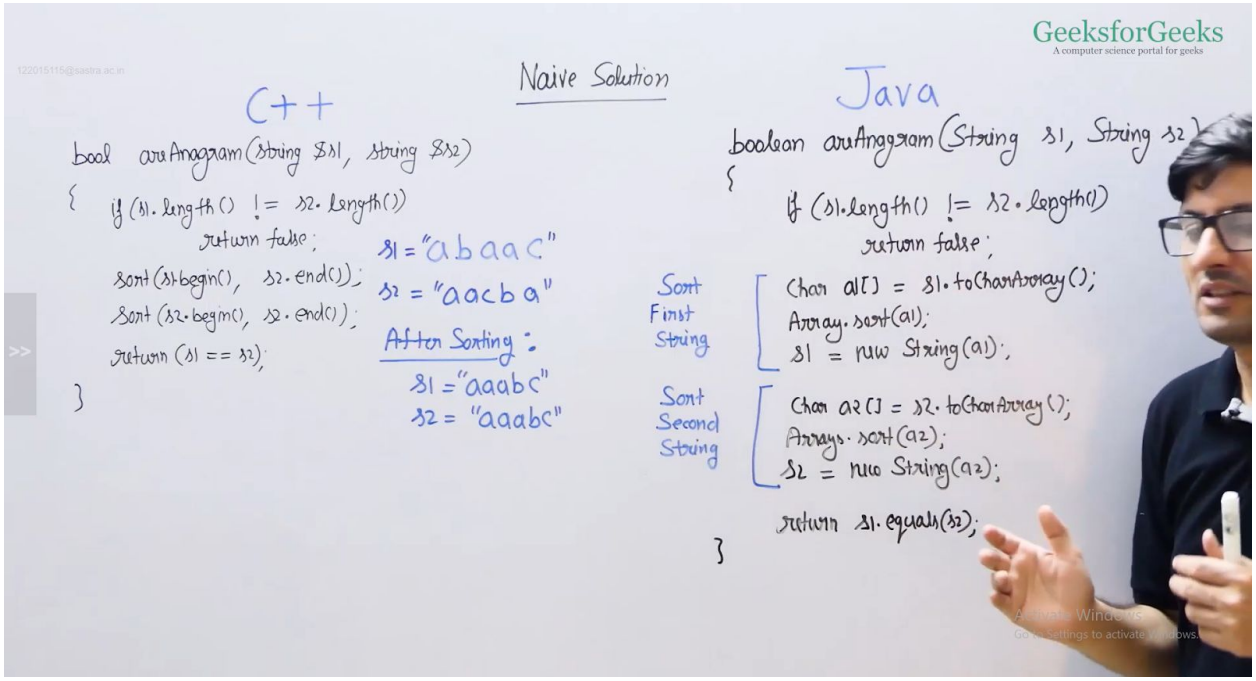


- Traversing a string using for and foreach loop

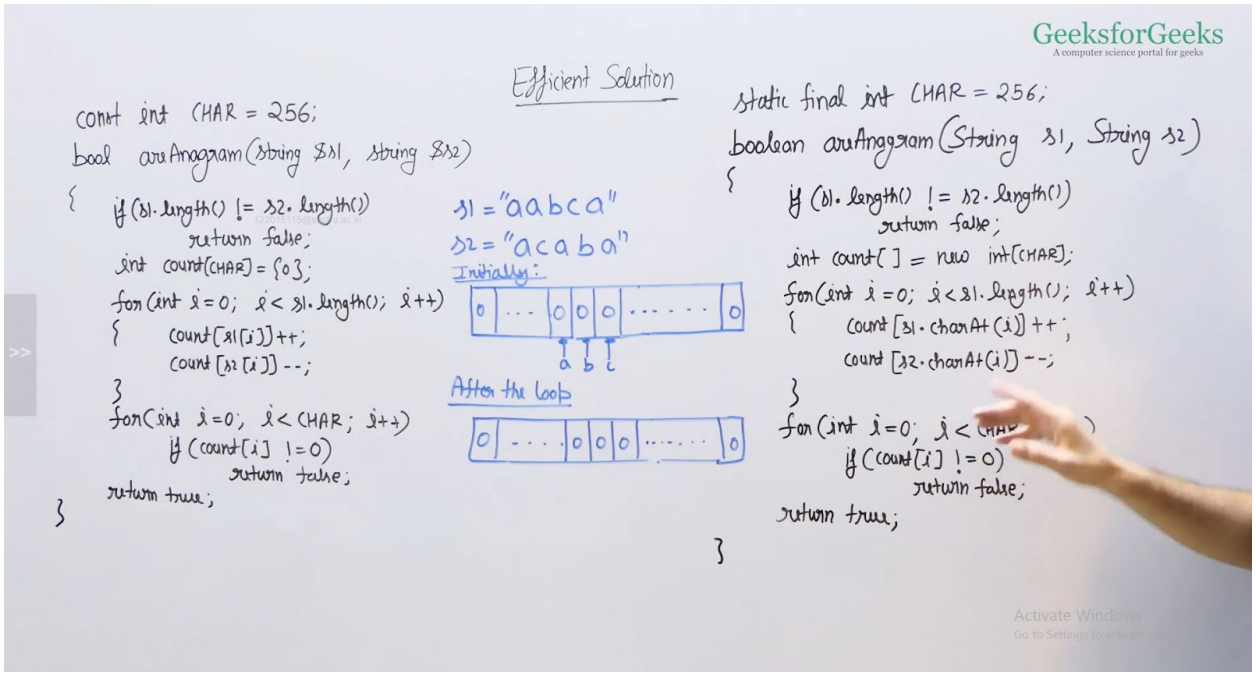
Check for Anagram

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





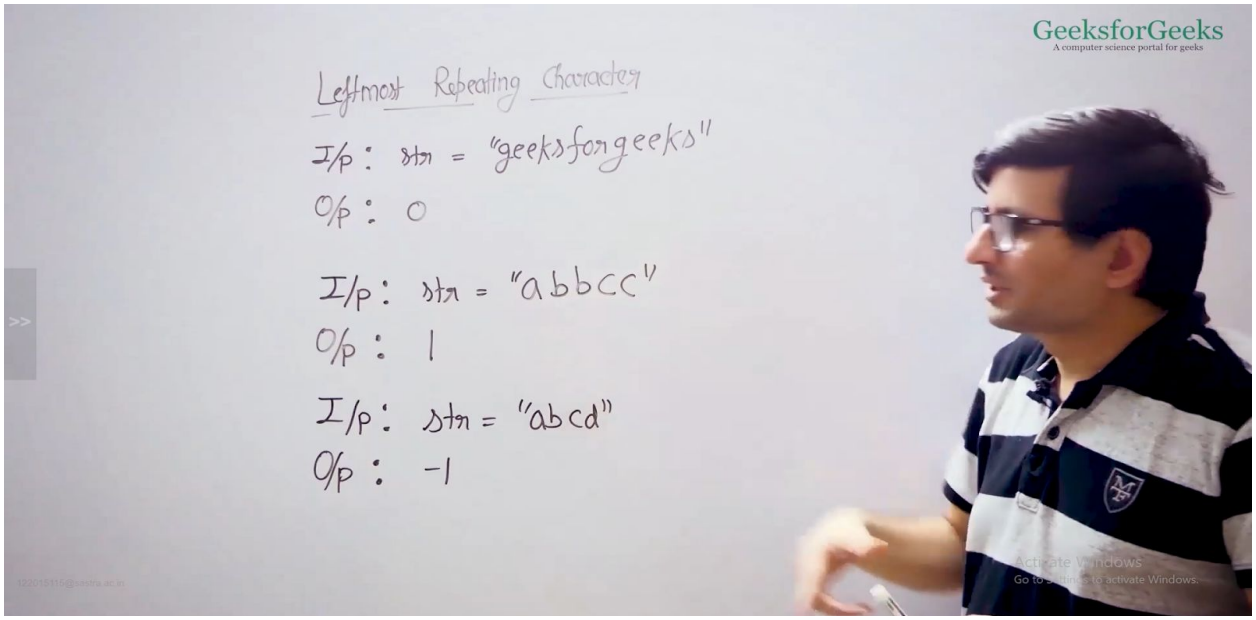
- Time Complexity : O(n log n)
- Space Complexity: O(1)

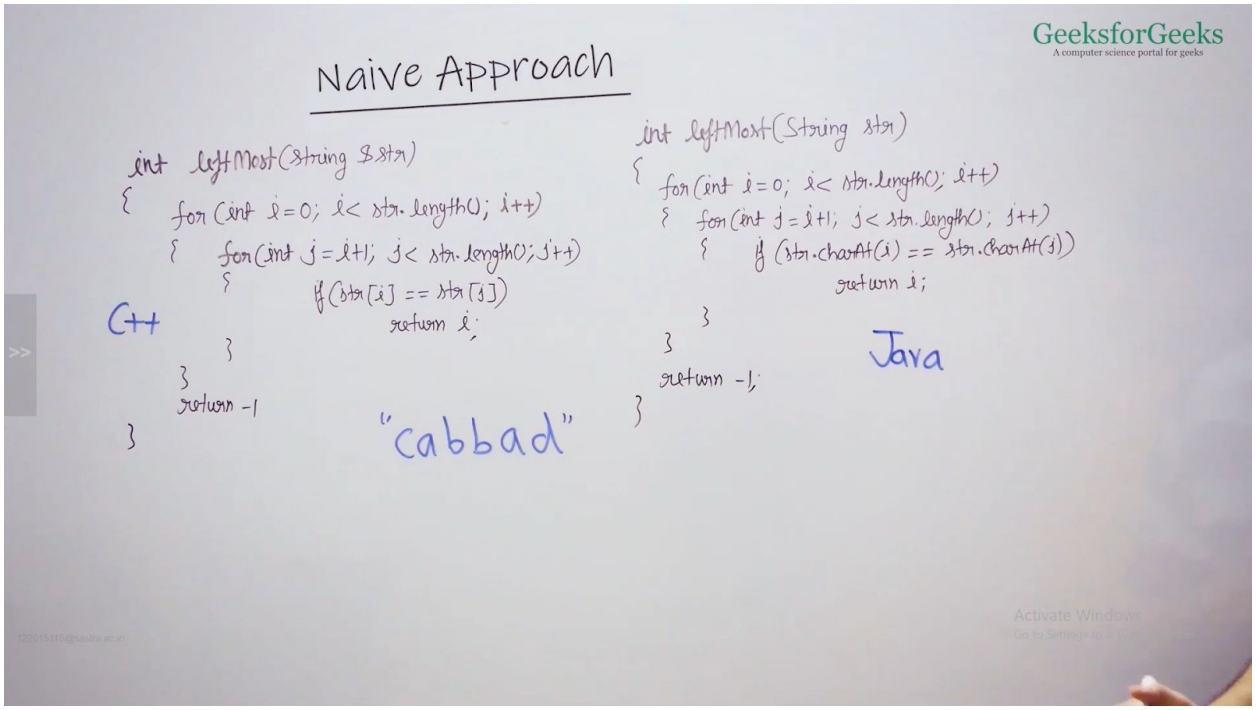


- Time : O(n + CHAR)
- Space : O(CHAR)

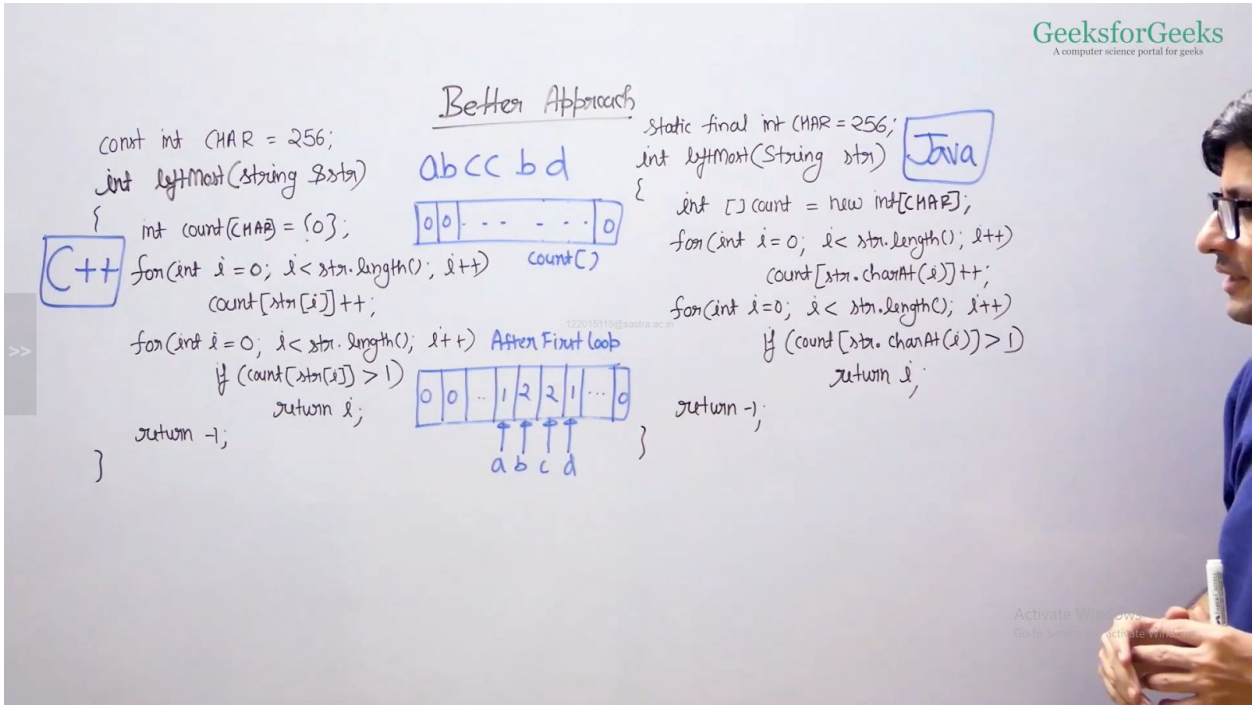
LeftMost Repeating Character

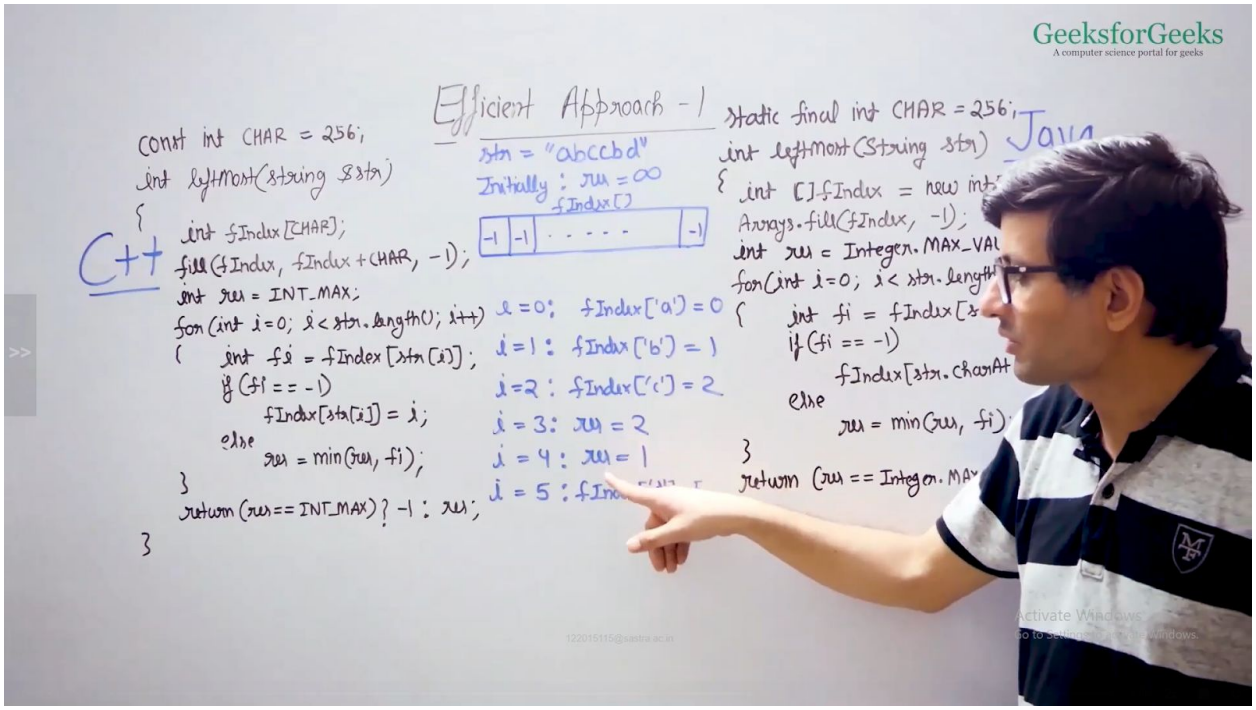
Given a string, the task is to find the first character (whose leftmost appearance is first) that repeats.

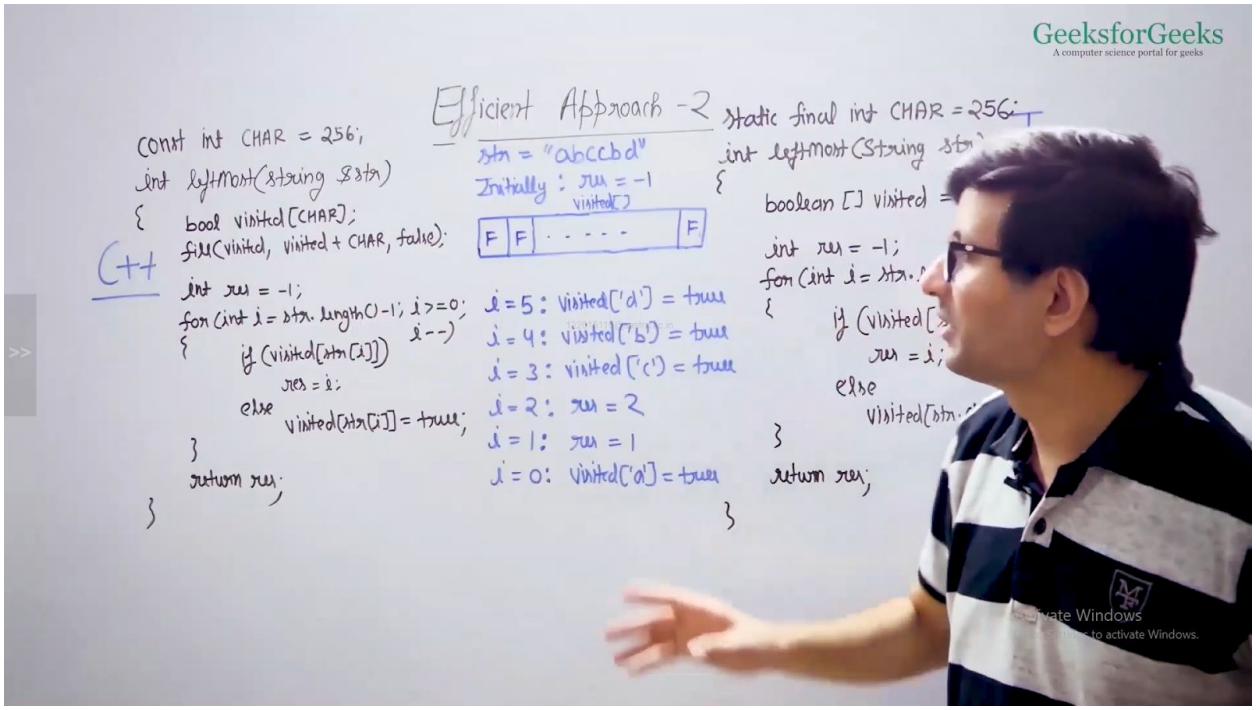




- Time: O(n^2)
- Space: O(1)



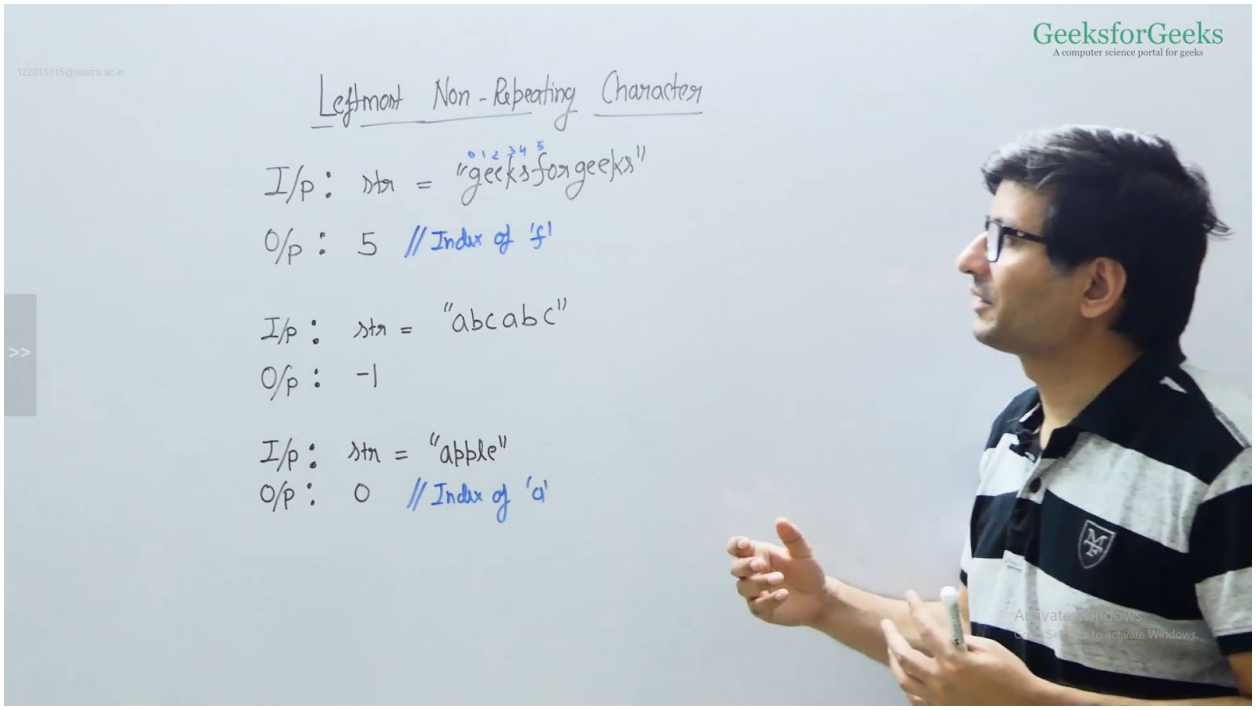


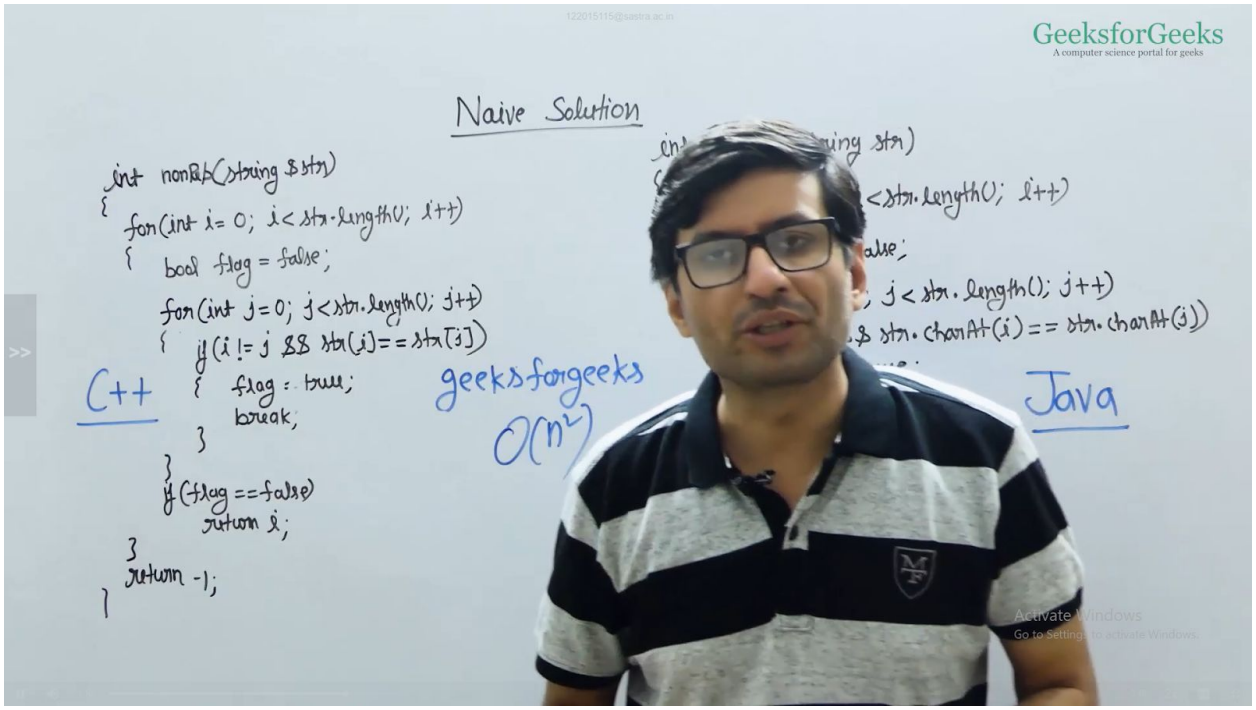


- Time: O(n) and Space: O(CHAR). 1st approach has more no of comparisons than 2nd

LeftMost Non-Repeating Element

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





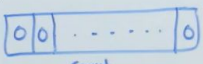
GeksforGeeks
A computer science portal for geeks

Better Solution (Two Traversals)

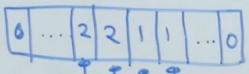
C++

```
const int CHAR = 256;
int nonRpt(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;
}
```

"abbcda"



After First Traversal



Java

```
static final int CHAR = 256;
int nonRpt(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;
}
```


GeksforGeeks
A computer science portal for geeks

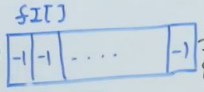
Efficient Solution (One Traversal)

C++

```
const int CHAR = 256;
int nonRpt(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 res = INT_MAX;
    for(int i=0; i<CHAR; i++)
        if (fI[i] >= 0)
            res = min(res, fI[i]);
    return (res == INT_MAX)? -1 : res;
}
```

"abbcbda"

fI[]



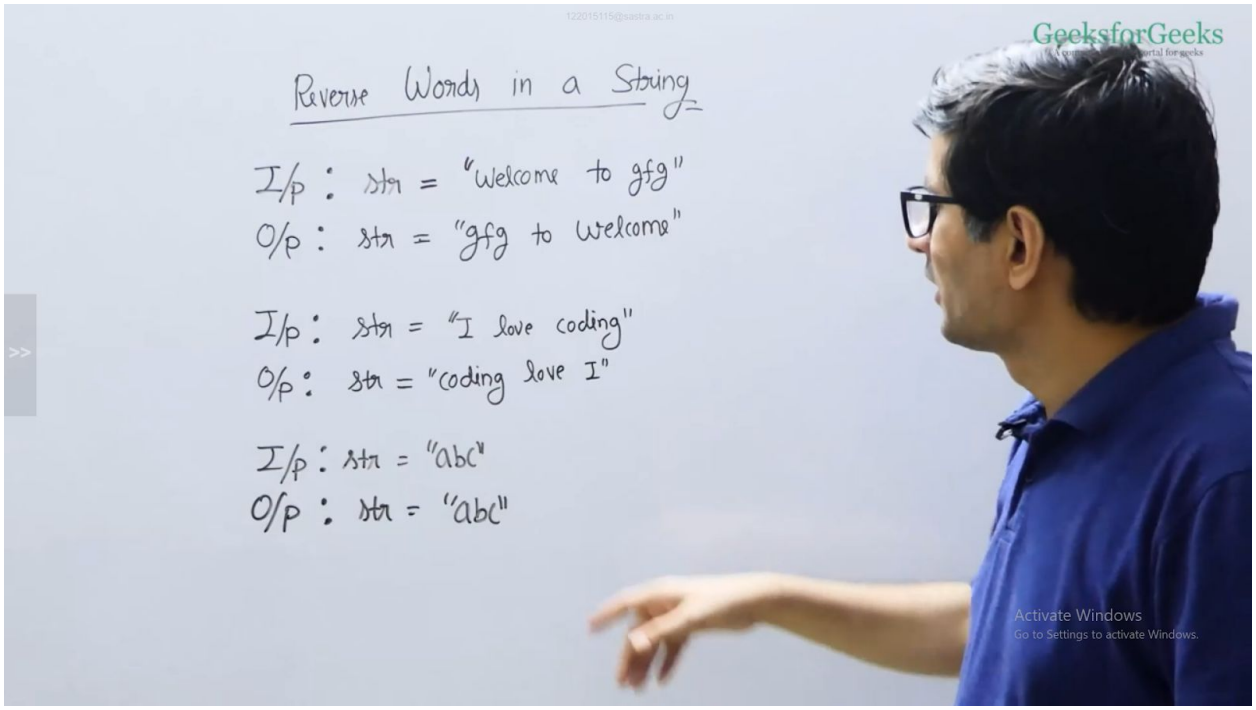
i=0: fI['a'] = 0
i=1: fI['b'] = 1
i=2: fI['b'] = -2
i=3: fI['c'] = 3
i=4: fI['b'] = -2
i=5: fI['d'] = 5
i=6: fI['a'] = -2

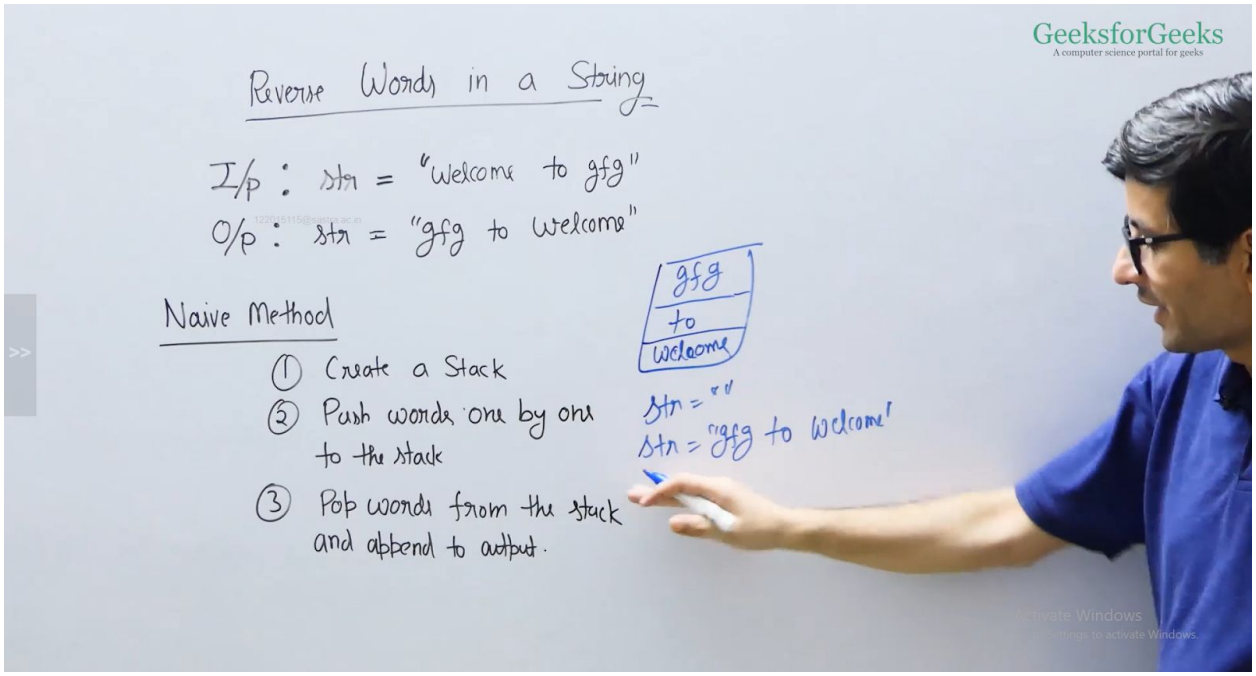
Java

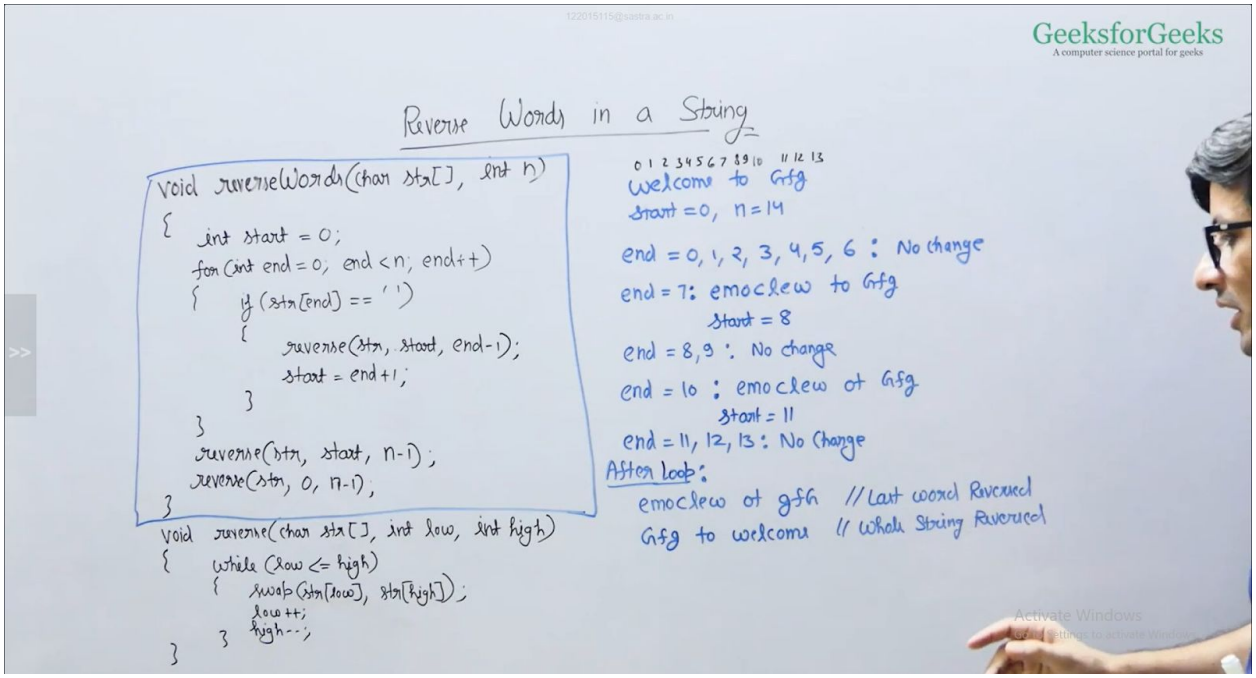
```
static final int CHAR = 256;
int nonRpt(String str)
{
    int []fI = new int[256];
    Arrays.fill(fI, -1);
    for(int i=0; i<str.length(); i++)
    {
        if (fI[str.charAt(i)] == -1)
            fI[str.charAt(i)] = i;
        else
            fI[str.charAt(i)] = -2;
    }
    int res = Integer.MAX_VALUE;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

Reverse Words in a String

Given a string, reverse its words.

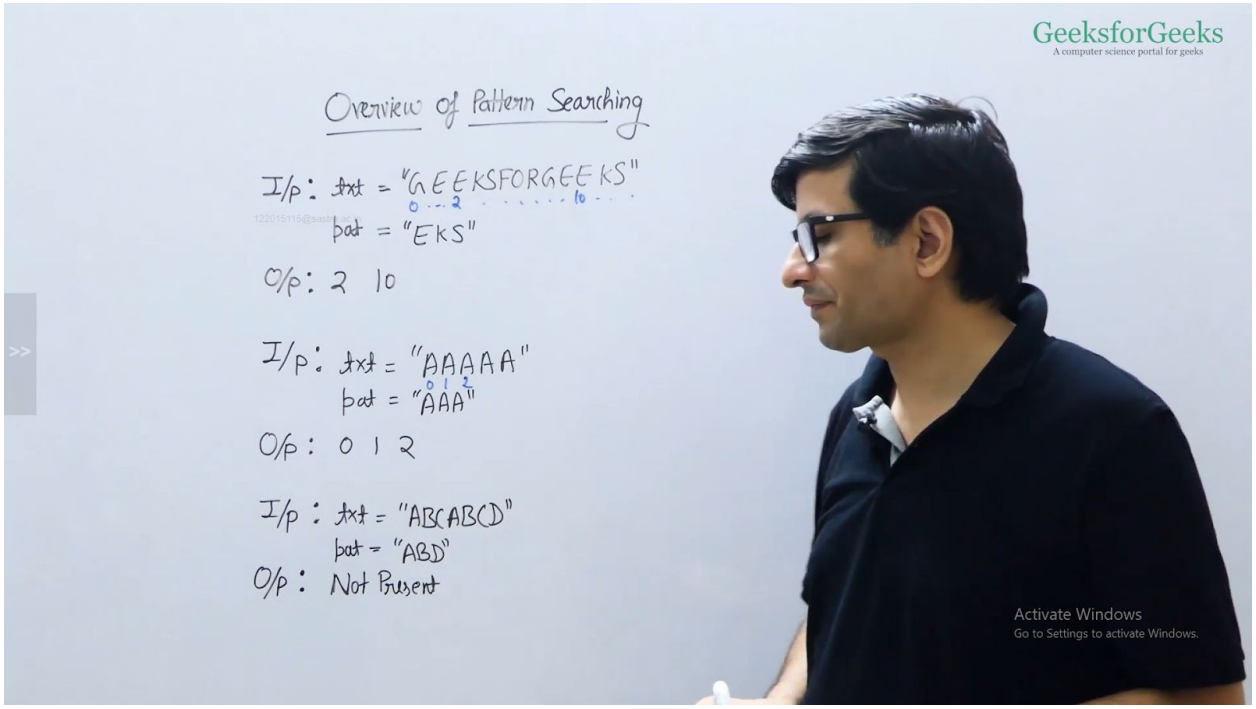




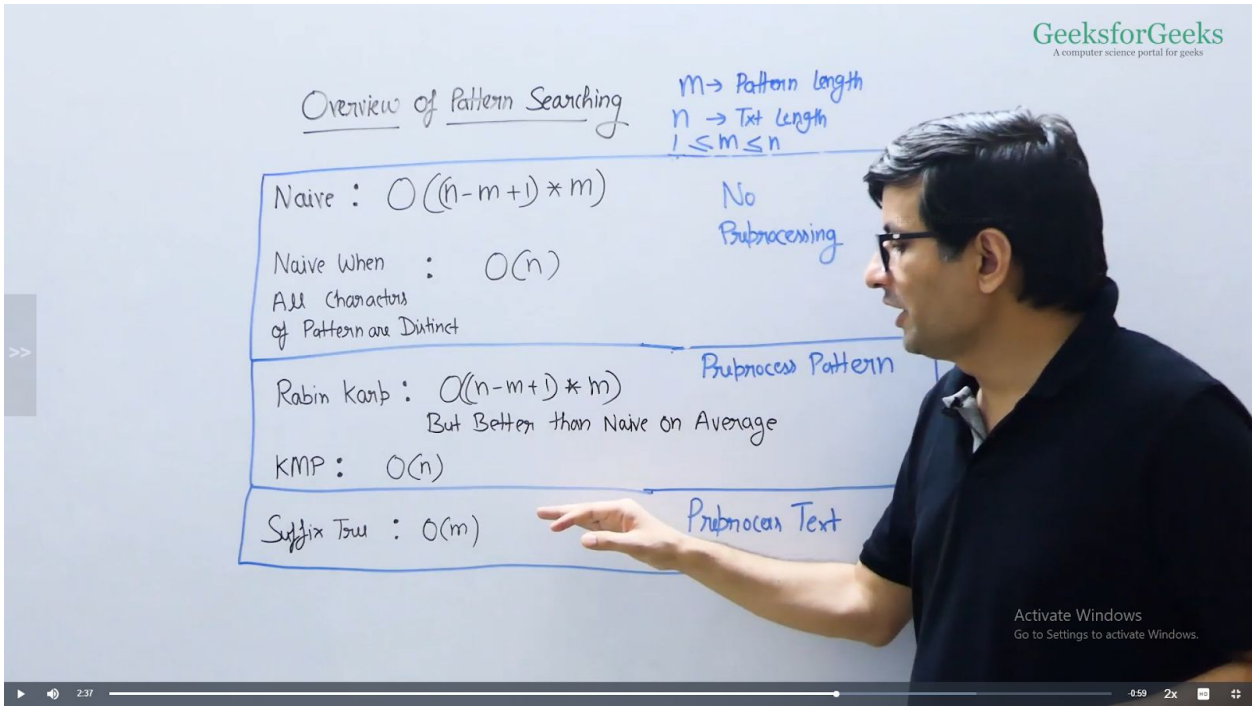


Pattern Searching

Overview:

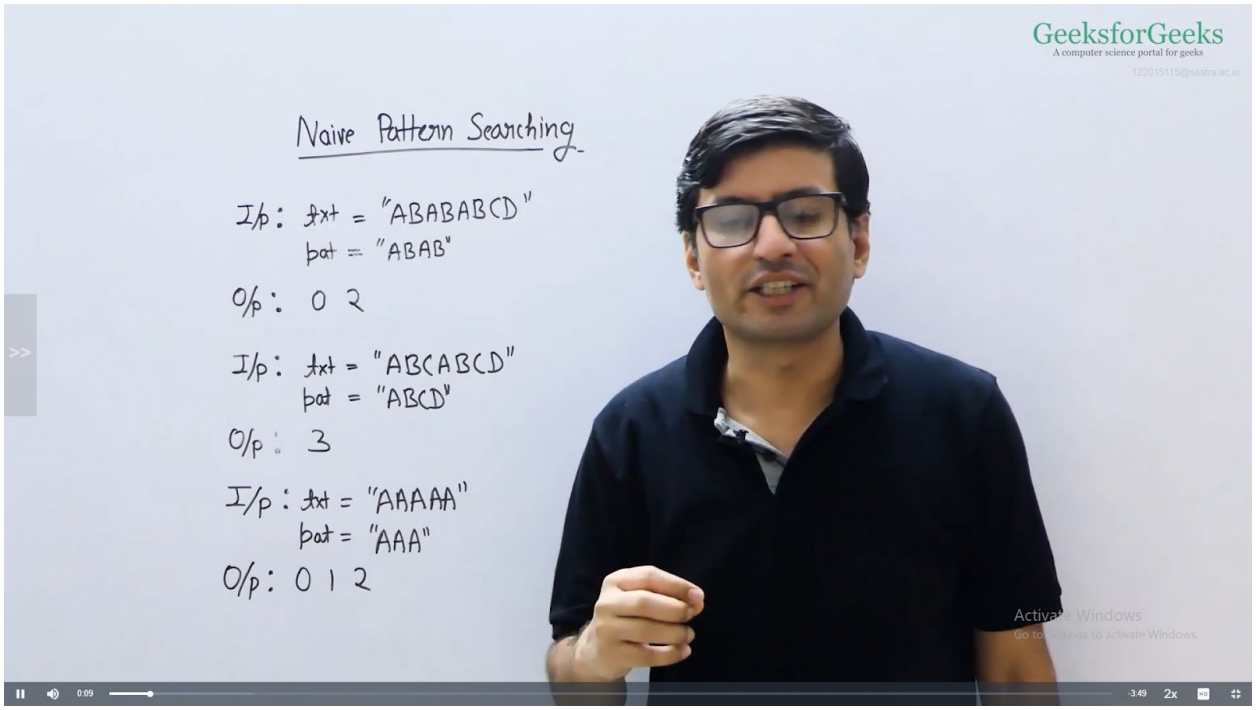


Given two strings text and pattern, we need to find occurrences of pattern in the string and return their indices.



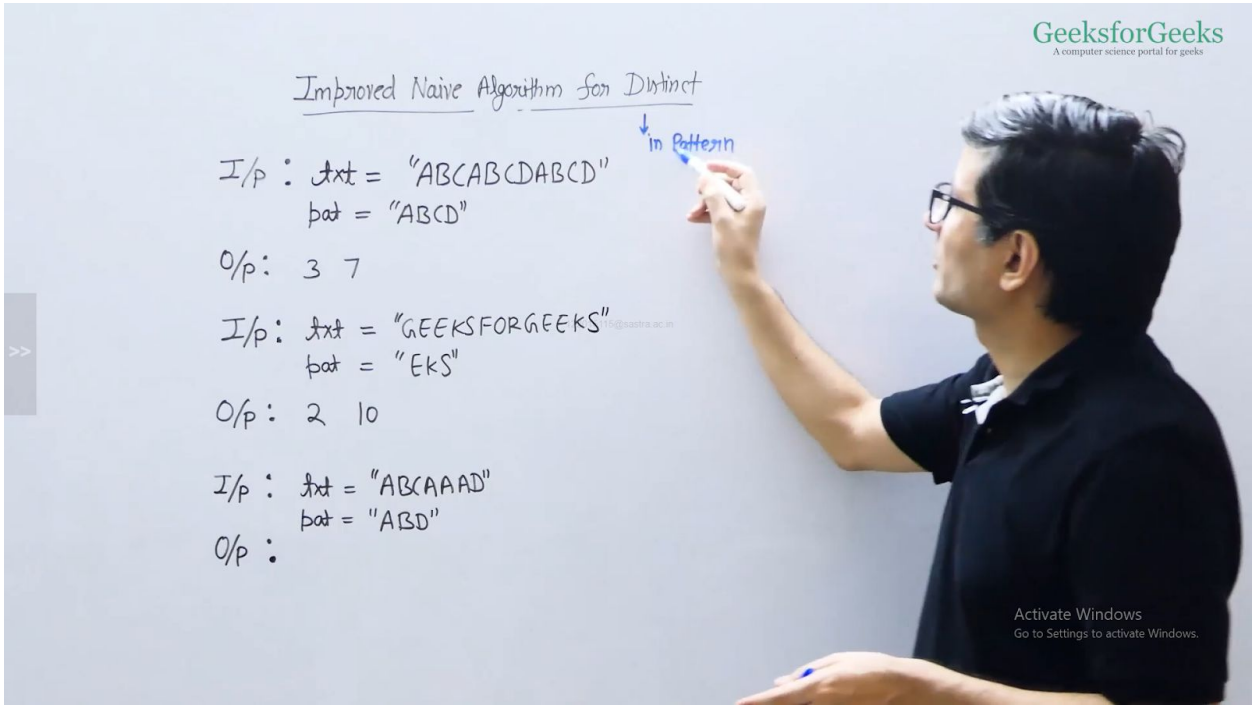
Naive Pattern Searching

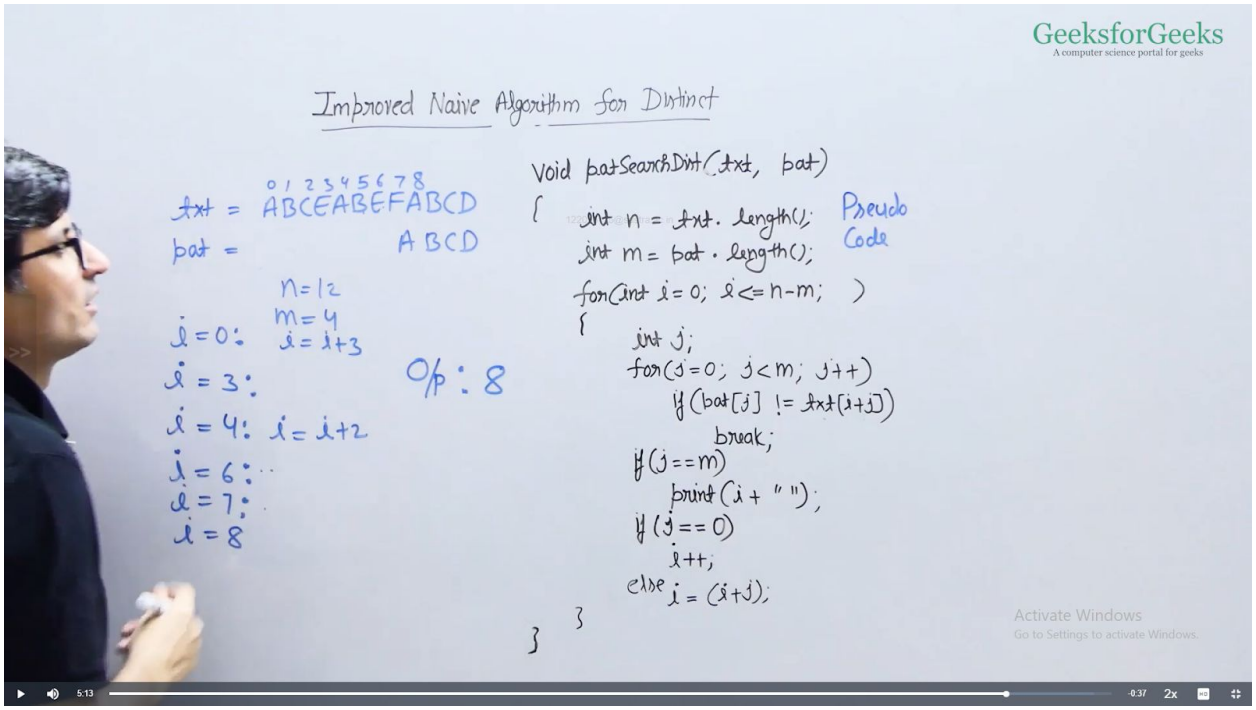
Given a pattern and a text, we need to print all occurrences of the pattern in the text. This takes about O((m+n-1)*m) time.



Improvised Pattern Matching with distinct elements in Pattern

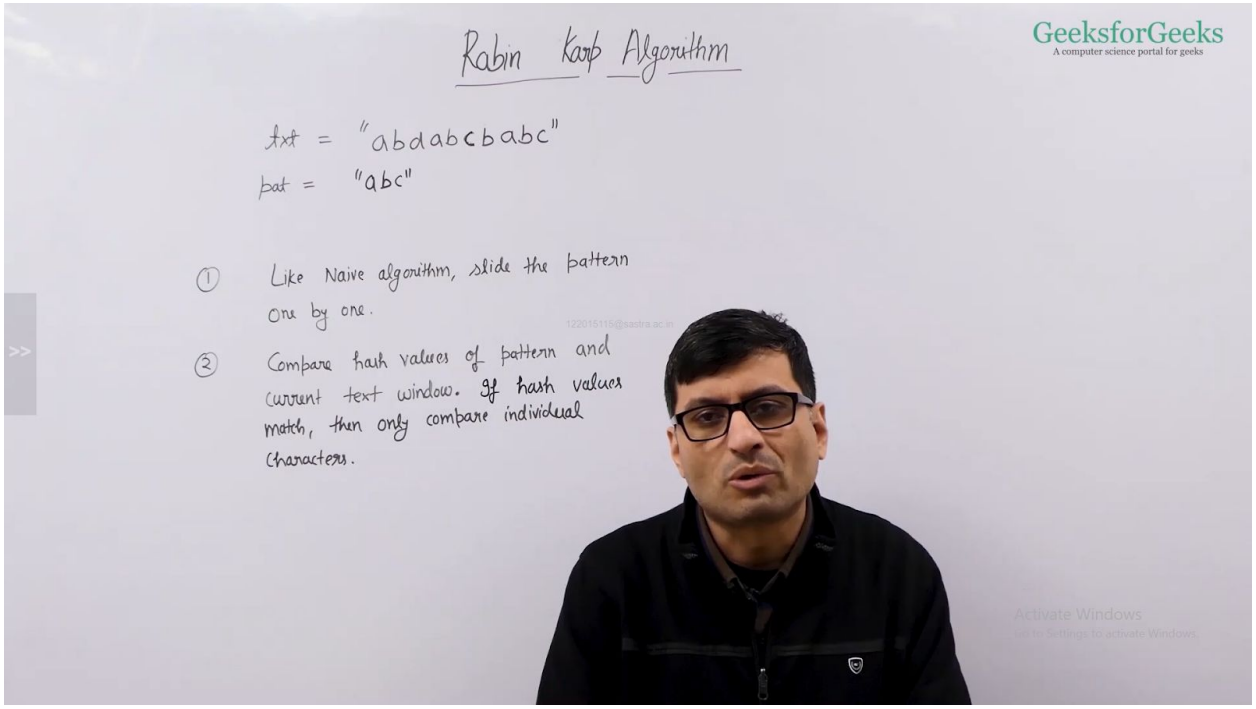
Given a pattern with distinct characters and a text, we need to print all occurrences of the pattern in the text. Time: O(n)





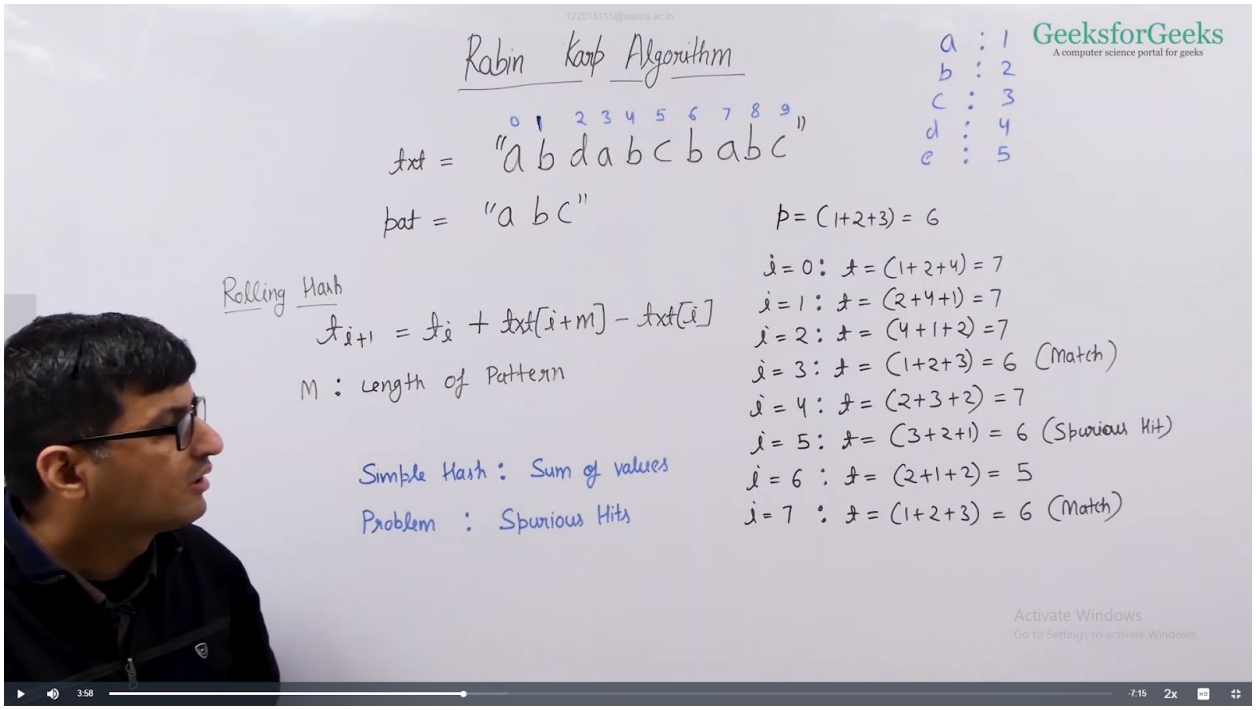
Rabin Karp Algorithm

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

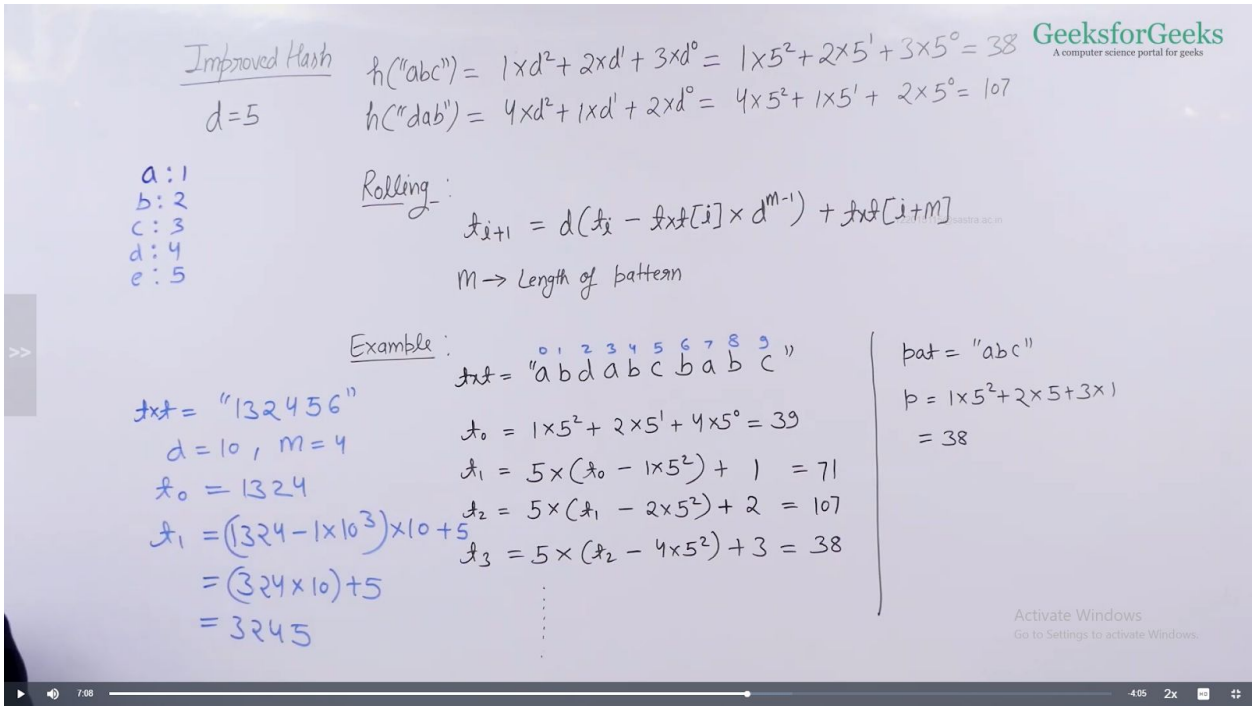


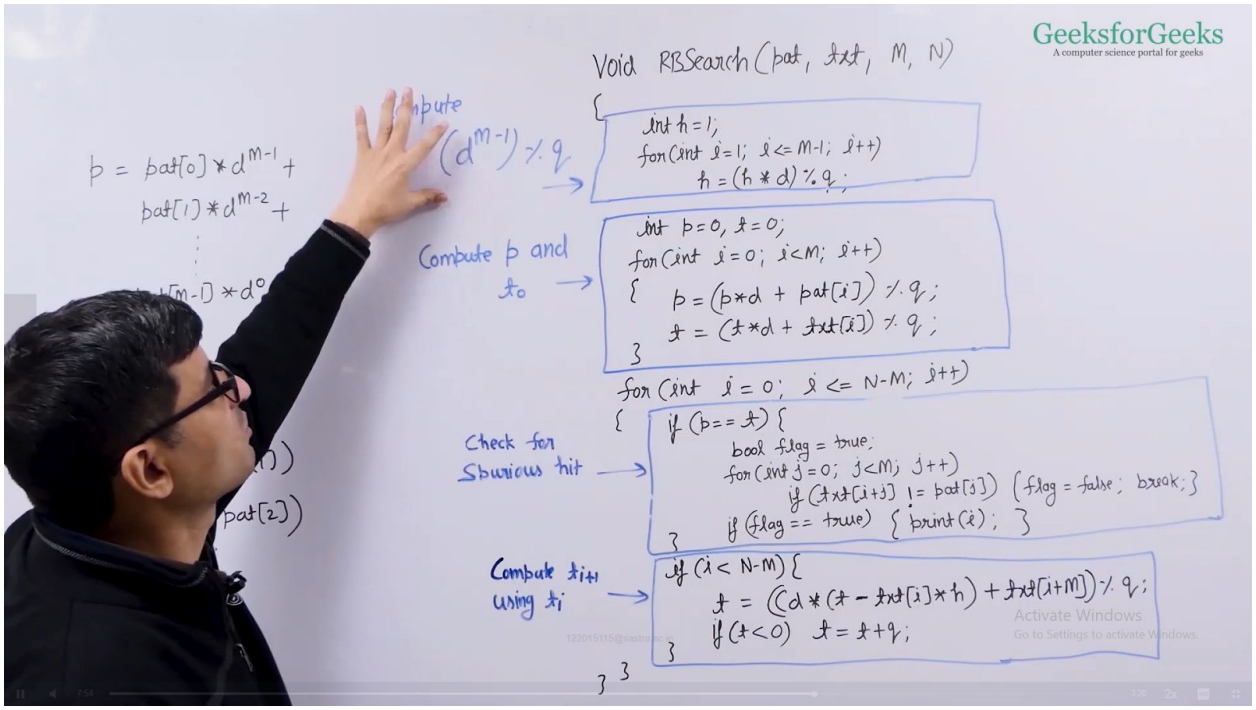
- This algorithm works better than the naive algorithm for large patterns

- 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



- 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



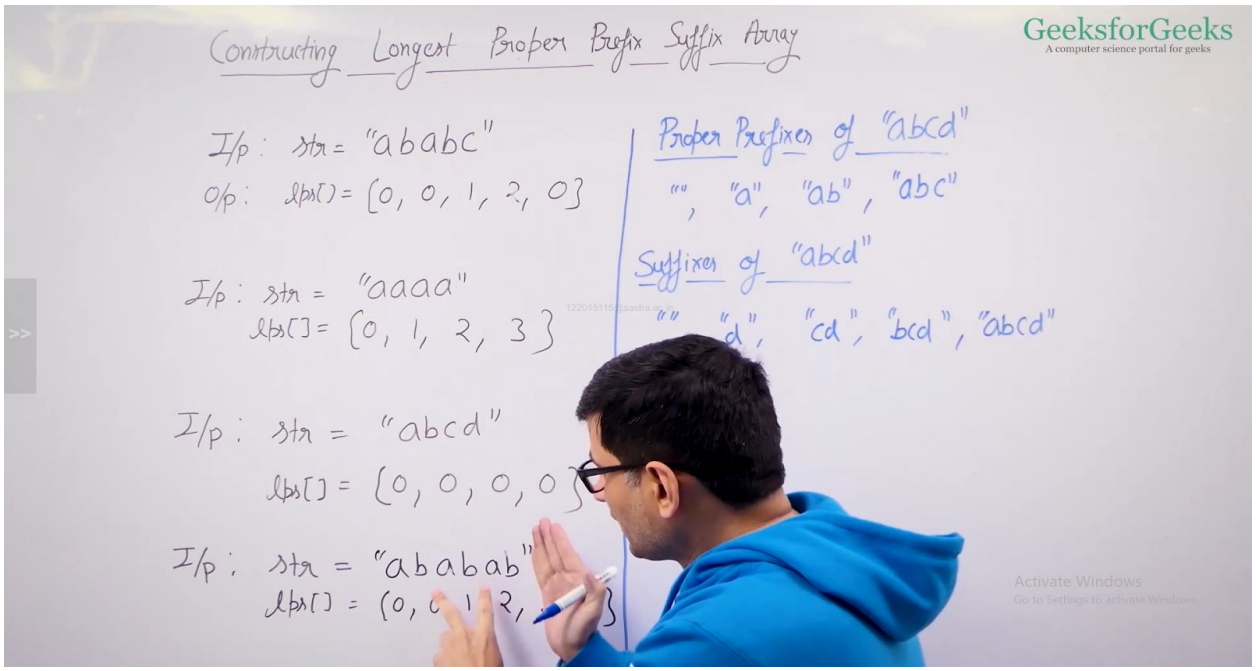


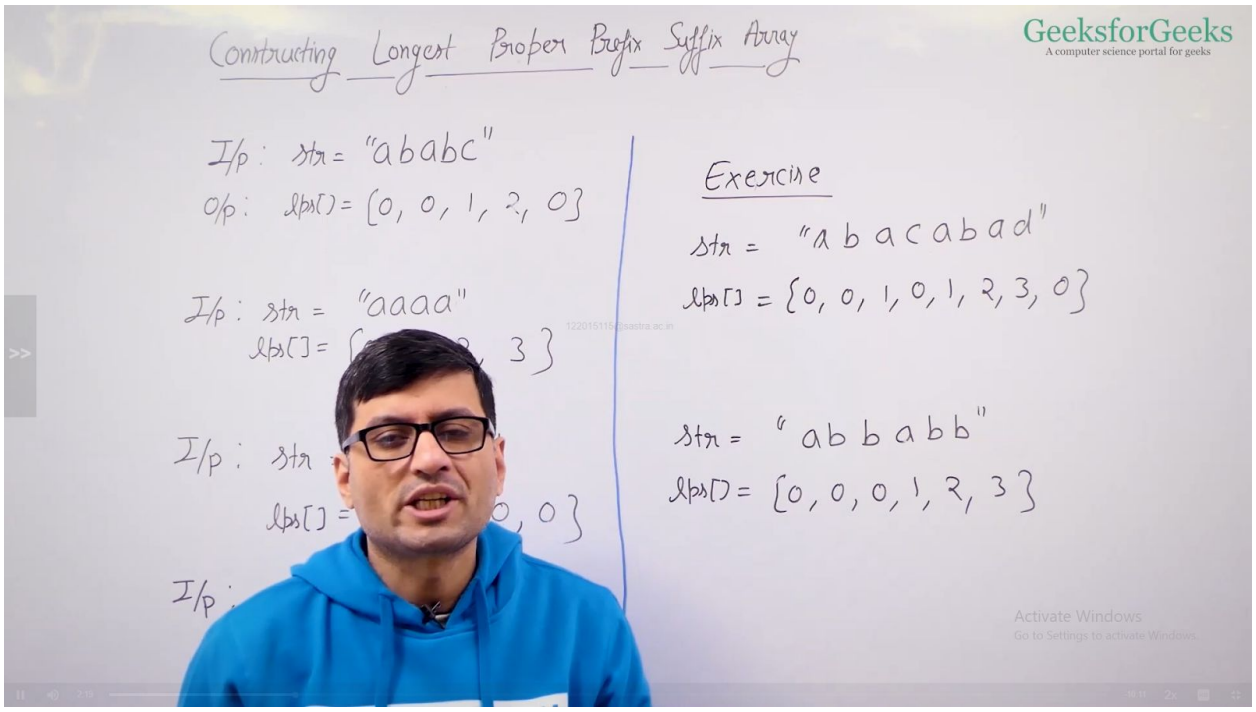
- The reason for doing computations under modulo q is that the computation of weightsum of ASCII values, the sum may go large and so we do modulo to store it in an int or long long int variable. Generally ‘q’ is a large prime number and it is taken as large as possible to reduce spurious hits

Construction of LPS Array:

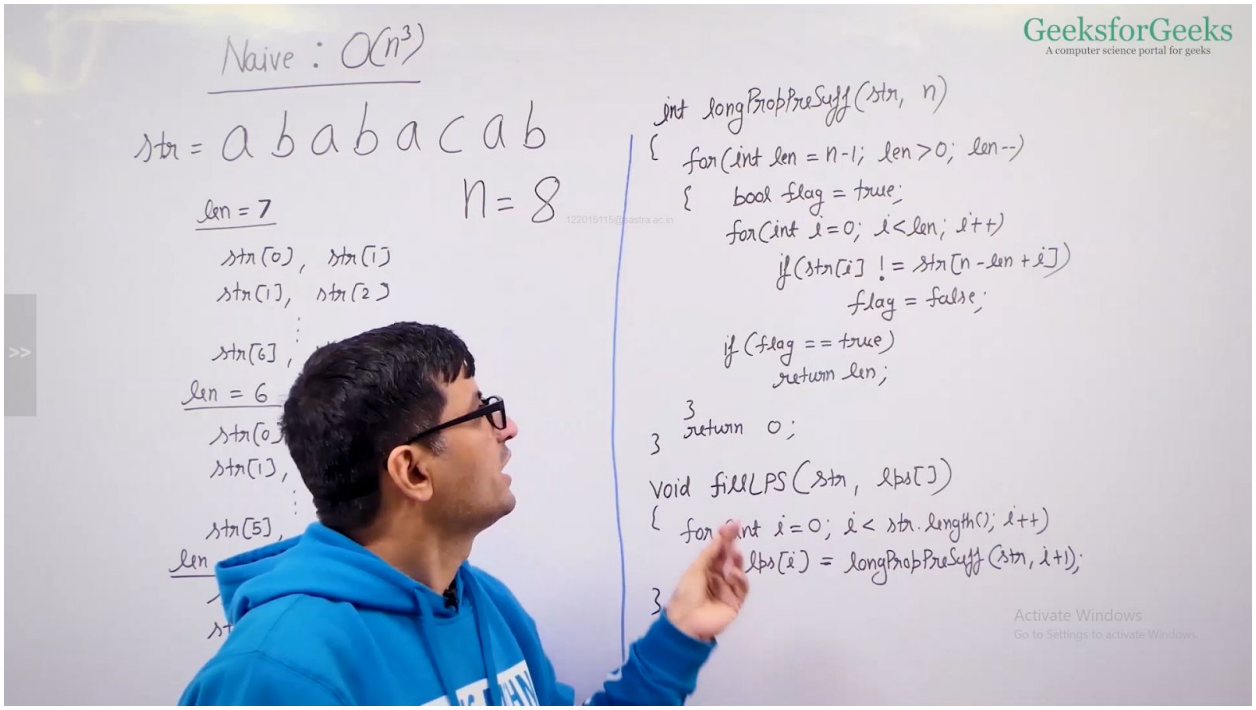
LPS => Longest Proper Prefix Suffix Array

- 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

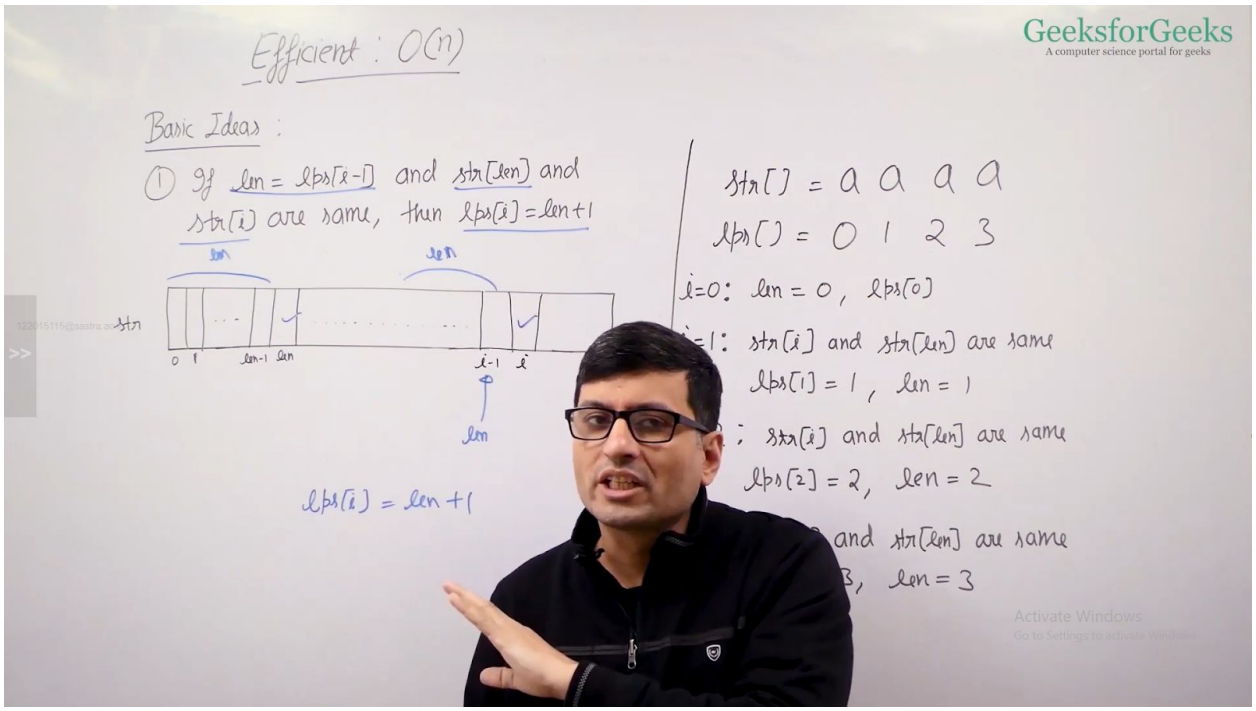




Naive Solution:O(n^3)



Optimized Solution: O(n)

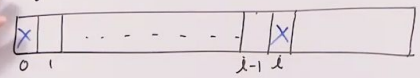


Efficient O(n)

Basic Ideas

(2) If str[i] and str[len] are not same.

(a) If len == 0

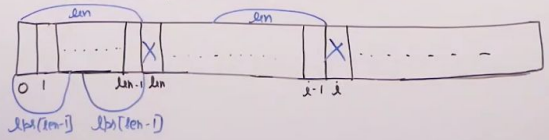


lps[i] = 0

(b) Else, we recursively apply lps[]

len = lps[len - 1]

then compare str[i] with str[len]



Implementation of O(n) algorithm:

```
    0 1 2 3 4 5 6 7 8
str = A A A B A A A A C
```

len = 0, lps[0] = 0

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

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

i = 3 : lps[

str[i]

again str[i]

len =

lps[

i = 4 :

i = 5 :

i =

```
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

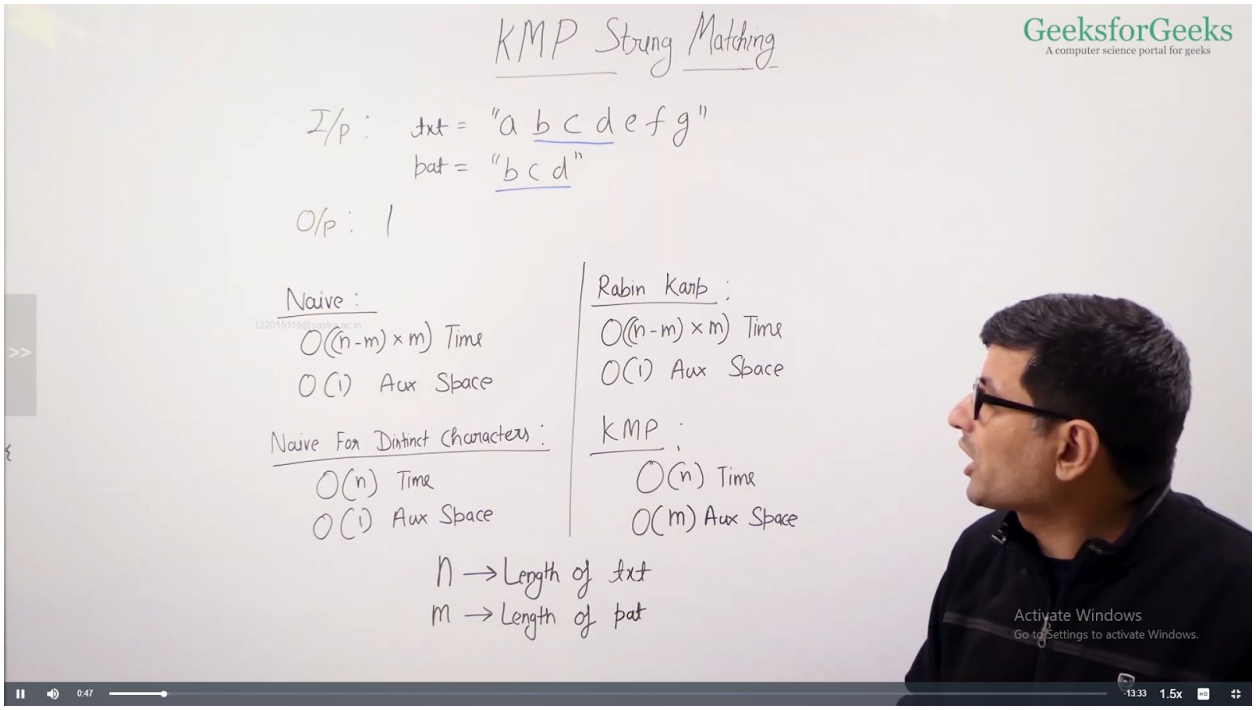
len = lps[1] = 1

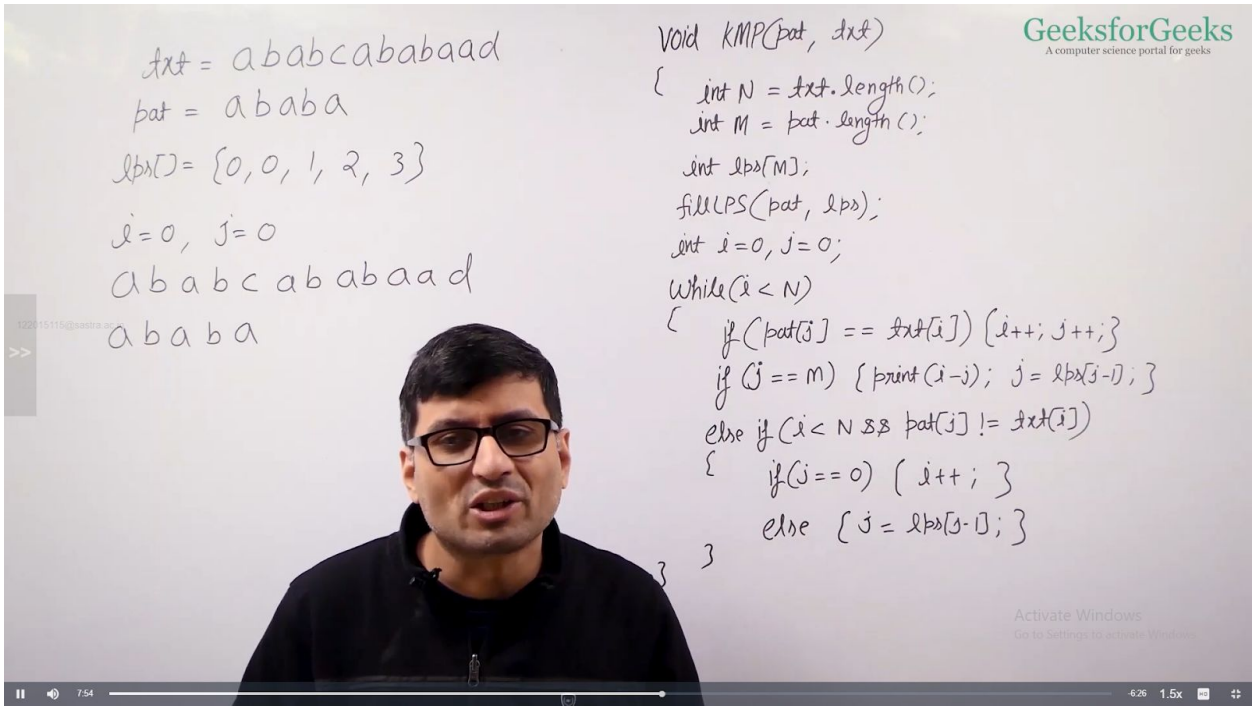
len = lps[0] = 0

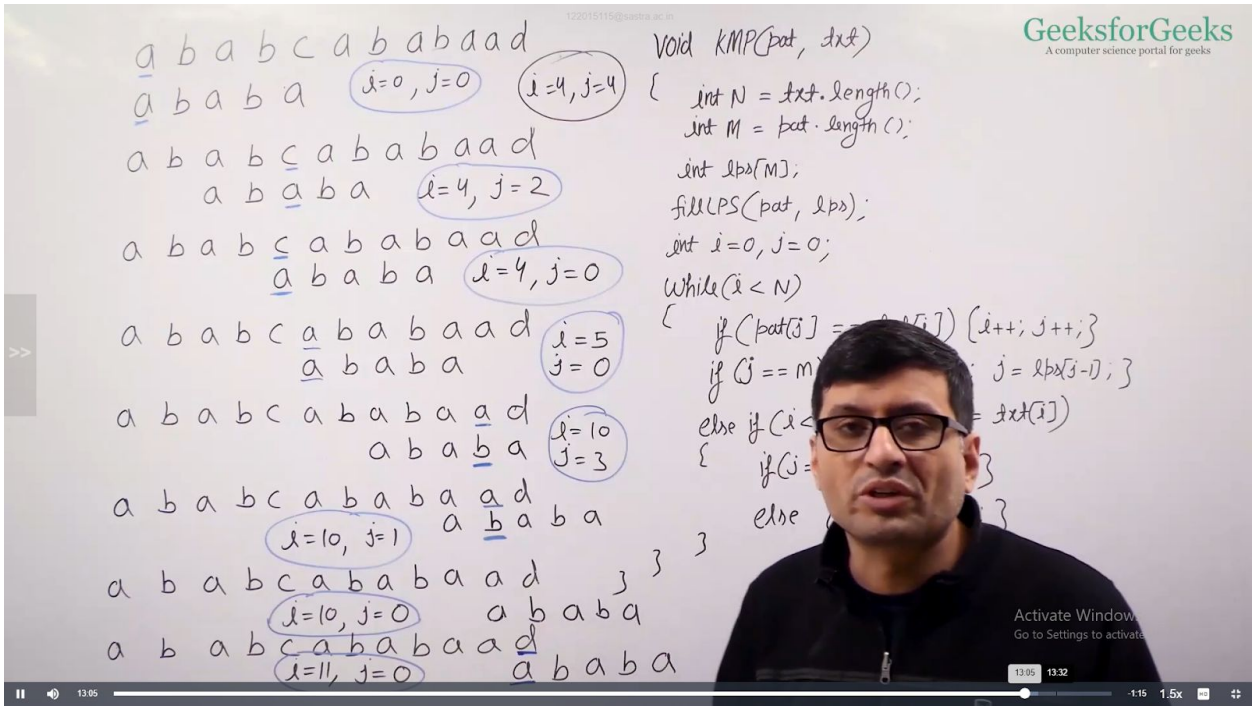
lps[8] = 0

KMP Algorithm

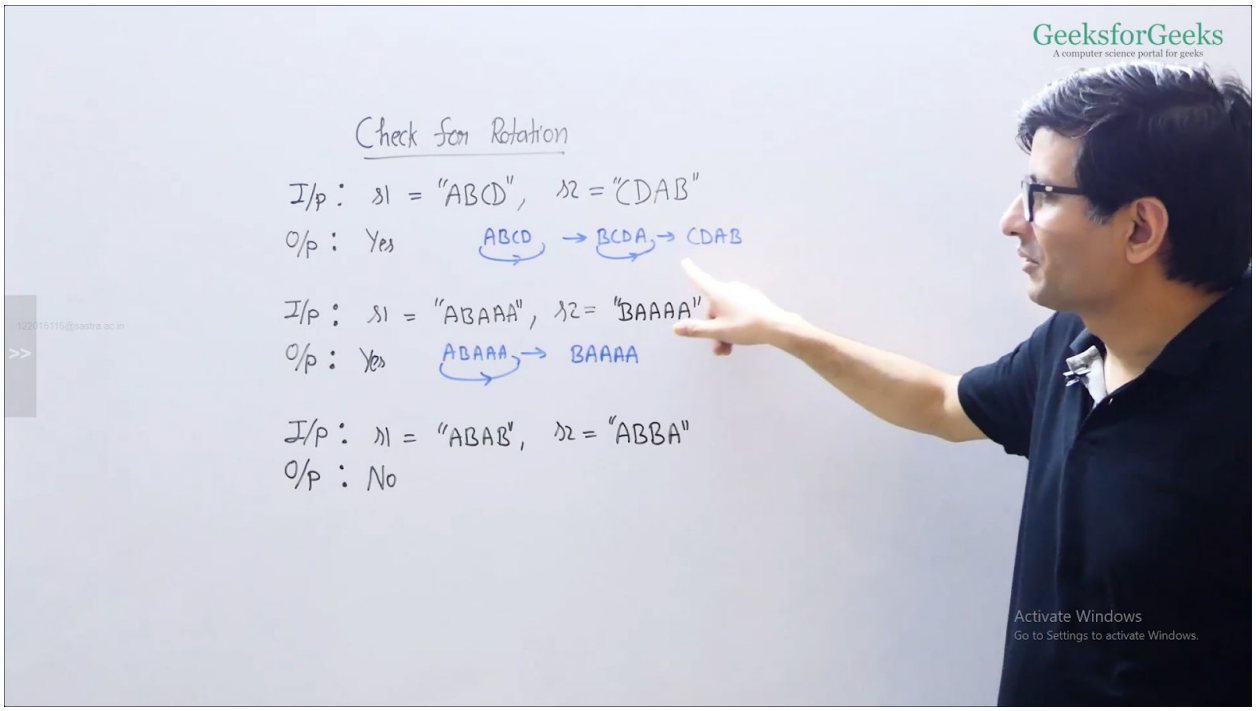
- For using KMP algorithm, we should know to construct LPS(Longest properPrefix Suffix) array







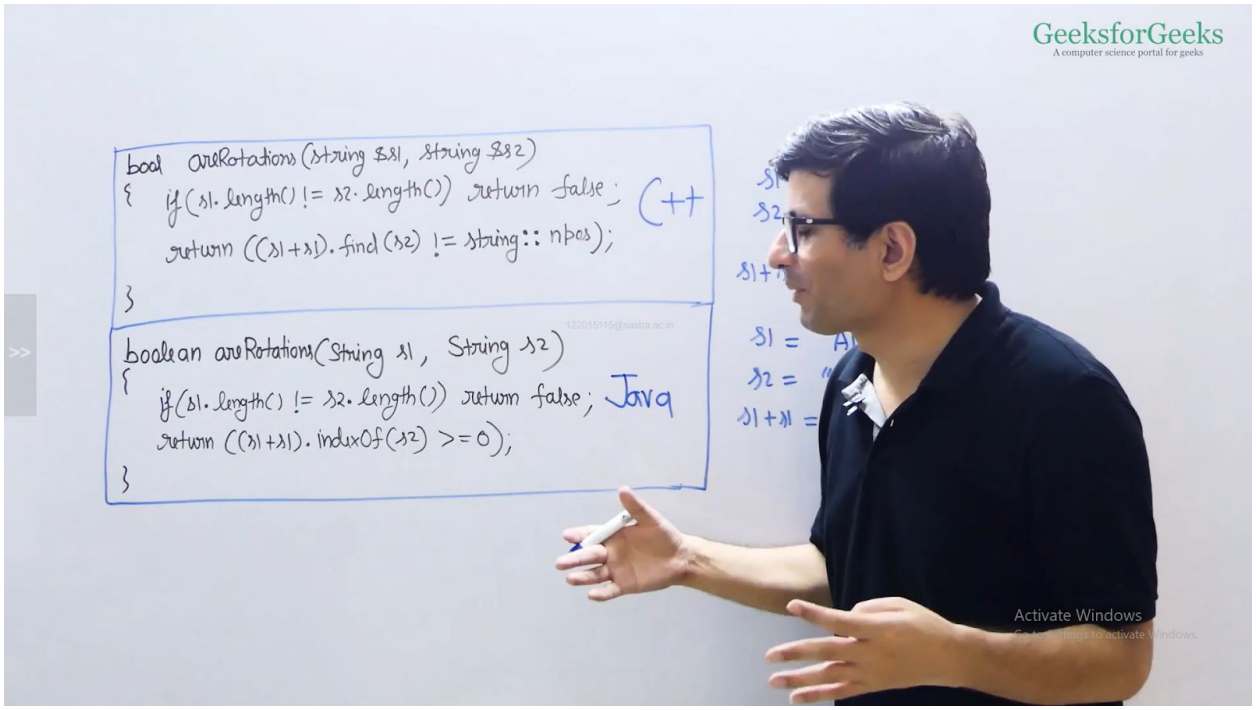
Check if Strings are Rotations



Naive Approach: O(n^2)

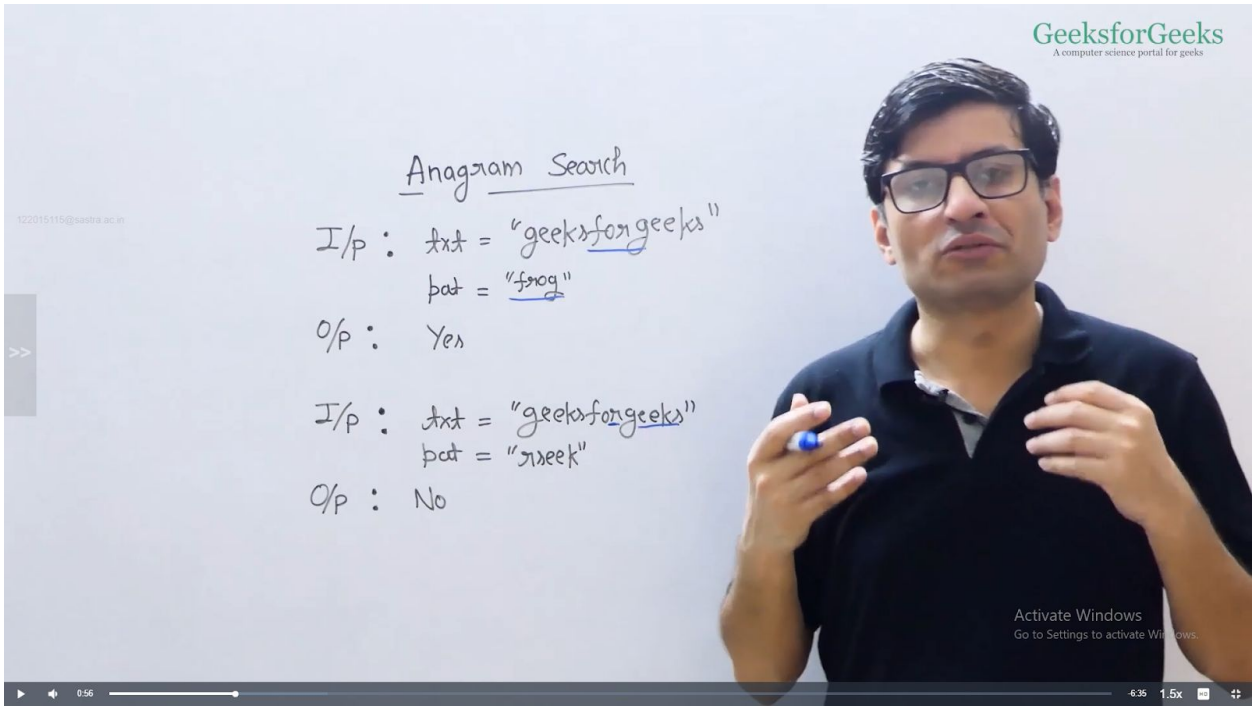
```
string lrotate(string& str, int k)
{
    string result;
    for(int i = k; i < str.length(); i++)
        result += str[i];
    for(int i = 0; i < k; i++)
        result += str[i];
    return result;
}
bool checkRotation(string& str1, string& str2)
{
    //Naive approach: O(n^2)
    for(int i = 0;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

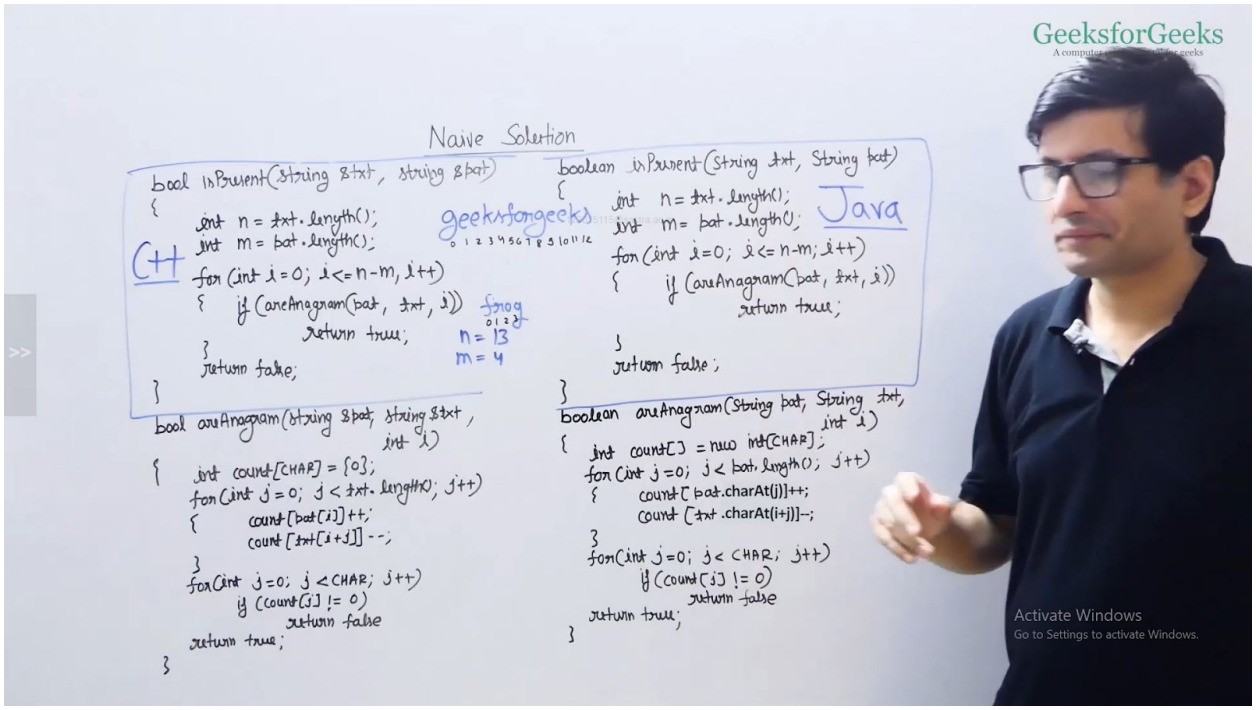
Efficient approach: O(n)

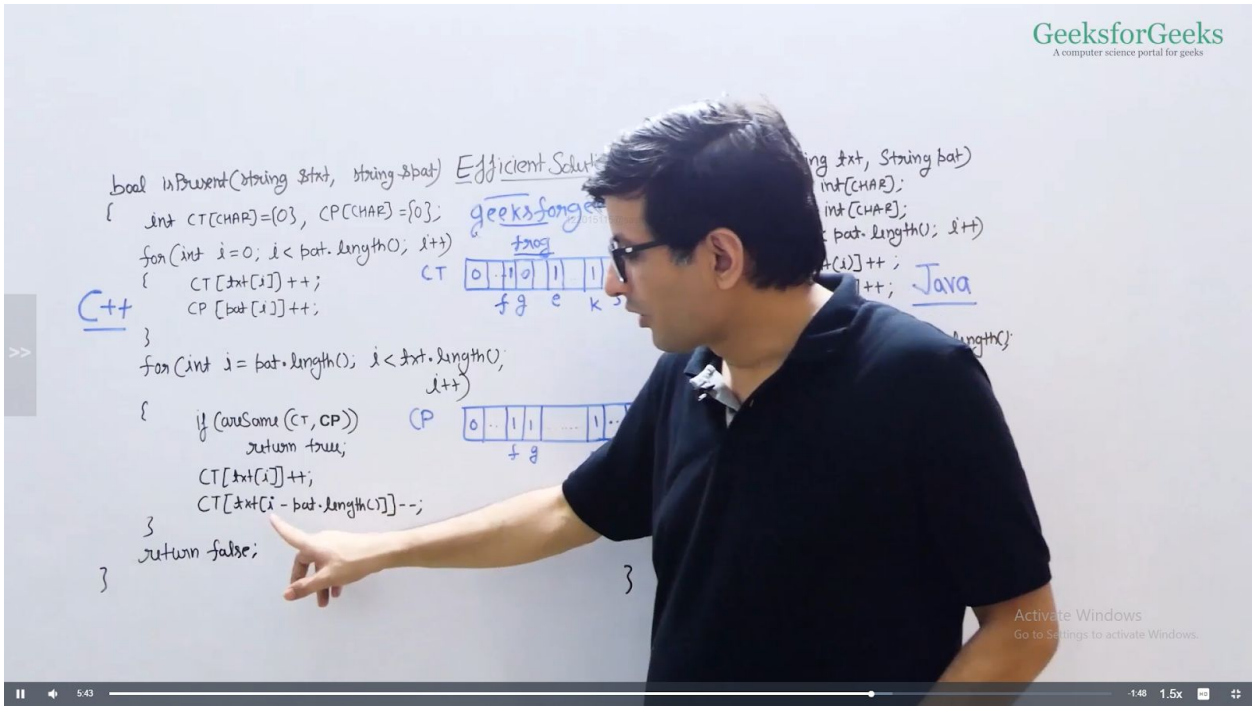


Anagram Search

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







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.

```
// return the smallest window in S with all the characters of P
// if no such window exists, return "-1"
string smallestWindow (string s, string p){
    // Your code here

    int m = p.length();
    int n = s.length();
    if(m > n)
        return "-1";
    int hash_pat[256] = {0};
    int hash_txt[256] = {0};
    for(int i = 0; i < m; i++)
    {
        hash_pat[p[i]]++;
    }
    int start = 0;
    int start_index = -1;
    int counter = 0;
    int min_length = INT_MAX;
    for(int i = 0; i < n; i++)
    {
        hash_txt[s[i]]++;
        if(hash_txt[s[i]] <= hash_pat[s[i]])
        {
            counter++;
        }
        if(counter == m)
        {
            while(hash_txt[s[start]] > hash_pat[s[start]] || hash_pat[s[start]] == 0)
            {
                if(hash_txt[s[start]] > hash_pat[s[start]])
                    hash_txt[s[start]]--;
                start++;
            }
            int len_window = i - start + 1;
            if(min_length > len_window)
            {
                min_length = len_window;
                start_index = start;
            }
        }
    }
    if(start_index == -1)
        return "-1";
    return s.substr(start_index, min_length);
}
} // } Driver Code Ends
```