Remove duplicates from a given string

Difficulty Level : Easy ● Last Updated : 18 May, 2021

Given a string **S**, the task is to remove all the duplicates in the given string.

Below are the different methods to remove duplicates in a string.

METHOD 1 (Simple)

C++

```
// CPP program to remove duplicate character
// from character array and print in sorted
// order
#include <bits/stdc++.h>
using namespace std;

char *removeDuplicate(char str[], int n)
{
   // Used as index in the modified string
   int index = 0;
```

Related Articles

```
      int j;
      for (j=0; j<i; j++)
         if (str[i] == str[j])
            break;

      // If not present, then add it to
      // result.
      if (j == i)
         str[index++] = str[i];
   }

   return str;
}
```

```
    char str[]= "geeksforgeeks";
    int n = sizeof(str) / sizeof(str[0]);
    cout << removeDuplicate(str, n);
    return 0;
}
```

Java

```
// Java program to remove duplicate character
// from character array and print in sorted
// order
import java.util.*;

class GFG
{
    static String removeDuplicate(char str[], int n)
    {
        // Used as index in the modified string
        int index = 0;

        // Traverse through all characters
        for (int i = 0; i < n; i++)
        {

            // Check if str[i] is present before it
            int j;
            for (j = 0; j < i; j++)
            {
                if (str[i] == str[j])
                {
                    break;
                }
            }

            // If not present, then add it to
            // result.
            if (j == i)
            {
                str[index++] = str[i];
            }
        }
        return String.valueOf(Arrays.copyOf(str, index));
    }

    // Driver code
    public static void main(String[] args)
    {
        char str[] = "geeksforgeeks".toCharArray();
        int n = str.length;
        System.out.println(removeDuplicate(str, n));
    }
}
```

Python3

```
# Python3 program to remove duplicate character
# from character array and prin sorted
# order
def removeDuplicate(str, n):

    # Used as index in the modified string
    index = 0

    # Traverse through all characters
    for i in range(0, n):

        # Check if str[i] is present before it
        for j in range(0, i + 1):
            if (str[i] == str[j]):
                break

        # If not present, then add it to
        # result.
        if (j == i):
            str[index] = str[i]
            index += 1

    return "".join(str[:index])

# Driver code
str= "geeksforgeeks"
n = len(str)
print(removeDuplicate(list(str), n))

# This code is contributed by SHUBHAMSINGH10
```

C#

```
// C# program to remove duplicate character
// from character array and print in sorted
// order
using System;
using System.Collections.Generic;
class GFG
{
static String removeDuplicate(char []str, int n)
{
    // Used as index in the modified string
    int index = 0;

    // Traverse through all characters
    for (int i = 0; i < n; i++)
    {
```

```
            {
                if (str[i] == str[j])
                {
                    break;
                }
            }

            // If not present, then add it to
            // result.
            if (j == i)
            {
                str[index++] = str[i];
            }
        }
        char [] ans = new char[index];
        Array.Copy(str, ans, index);
        return String.Join("", ans);
    }

    // Driver code
    public static void Main(String[] args)
    {
        char []str = "geeksforgeeks".ToCharArray();
        int n = str.Length;
        Console.WriteLine(removeDuplicate(str, n));
    }
    }

    // This code is contributed by PrinciRaj1992
```

Output:

```
geksfor
```

Time Complexity : O(n * n)

Auxiliary Space : O(1)

Keeps order of elements same as input.

METHOD 2 (Use BST)

use set in c++ stl which implements a self balancing Binary Search Tree.

CPP

```
#include <bits/stdc++.h>
using namespace std;

char *removeDuplicate(char str[], int n)
{
    // create a set using string characters
    // excluding '\0'
    set<char>s (str, str+n-1);

    // print content of the set
    int i = 0;
    for (auto x : s)
       str[i++] = x;
    str[i] = '\0';

    return str;
}

// Driver code
int main()
{
   char str[]= "geeksforgeeks";
   int n = sizeof(str) / sizeof(str[0]);
   cout << removeDuplicate(str, n);
   return 0;
}
```

Output:


```
efgkors
```

Time Complexity: O(n Log n)

Does not keeps order of elements same as input, but prints them in sorted order.

METHOD 3 (Use Sorting)

Algorithm:

```
1) Sort the elements.
2) Now in a loop, remove duplicates by comparing the
    current character with previous character.
3)  Remove extra characters at the end of the resultant string.
```

Example:

```
Input string:  geeksforgeeks
1) Sort the characters
   eeeefggkkorss
2) Remove duplicates
    efgkorskkorss
3) Remove extra characters
     efgkors
```

Note that, this method doesn't keep the original order of the input string. For example, if we are to remove duplicates for geeksforgeeks and keep the order of characters same, then output should be geksfor, but above function returns efgkos. We can modify this method by storing the original order. METHOD 2 keeps the order same.

Implementation:

C++

```
// C++ prigram to remove duplicates, the order of
// characters is not maintained in this program
#include<bits/stdc++.h>
using namespace std;

/* Function to remove duplicates in a sorted array */
char *removeDupsSorted(char *str)
{
    int res_ind = 1, ip_ind = 1;

    /* In place removal of duplicate characters*/
    while (*(str + ip_ind))
    {
        if (*(str + ip_ind) != *(str + ip_ind - 1))
```

```
                res_ind++;
            }
            ip_ind++;
        }

        /* After above step string is efgkorskkorss.
           Removing extra kkorss after string*/
        *(str + res_ind) = '\0';

        return str;
    }

    /* Function removes duplicate characters from the string
       This function work in-place and fills null characters
       in the extra space left */
    char *removeDups(char *str)
    {
       int n = strlen(str);

       // Sort the character array
       sort(str, str+n);

       // Remove duplicates from sorted
       return removeDupsSorted(str);
    }

    /* Driver program to test removeDups */
    int main()
    {
      char str[] = "geeksforgeeks";
      cout << removeDups(str);
      return 0;
    }
```

C

```
// C++ program to remove duplicates, the order of
// characters is not maintained in this program
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
/* Function to remove duplicates in a sorted array */
char *removeDupsSorted(char *str);

/* Utitlity function to sort array A[] */
void quickSort(char A[], int si, int ei);

/* Function removes duplicate characters from the string
   This function work in-place and fills null characters
   in the extra space left */
char *removeDups(char *str)
{
  int len = strlen(str);
```

```
}

/* Function to remove duplicates in a sorted array */
char *removeDupsSorted(char *str)
{
  int res_ind = 1, ip_ind = 1;

  /* In place removal of duplicate characters*/
  while (*(str + ip_ind))
  {
    if (*(str + ip_ind) != *(str + ip_ind - 1))
    {
      *(str + res_ind) = *(str + ip_ind);
      res_ind++;
    }
    ip_ind++;
  }

  /* After above step string is efgkorskkorss.
     Removing extra kkorss after string*/
  *(str + res_ind) = '\0';

  return str;
}

/* Driver program to test removeDups */
int main()
{
  char str[] = "geeksforgeeks";
  printf("%s", removeDups(str));
  getchar();
  return 0;
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SORTING
    PURPOSE */
void exchange(char *a, char *b)
{
  char temp;
  temp = *a;
  *a   = *b;
  *b   = temp;
}

int partition(char A[], int si, int ei)
{
  char x = A[ei];
  int i = (si - 1);
  int j;

  for (j = si; j <= ei - 1; j++)
  {
    if (A[j] <= x)
    {
      i++;
```

```
  exchange (&A[i + 1], &A[ei]);
  return (i + 1);
}

/* Implementation of Quick Sort
A[] --> Array to be sorted
si  --> Starting index
ei  --> Ending index
*/
void quickSort(char A[], int si, int ei)
{
  int pi;    /* Partitioning index */
  if (si < ei)
  {
    pi = partition(A, si, ei);
    quickSort(A, si, pi - 1);
    quickSort(A, pi + 1, ei);
  }
}
```

Java

```
// Java program to remove duplicates, the order of
// characters is not maintained in this program

import java.util.Arrays;

public class GFG
{
    /* Method to remove duplicates in a sorted array */
    static String removeDupsSorted(String str)
    {
        int res_ind = 1, ip_ind = 1;

        // Character array for removal of duplicate characters
        char arr[] = str.toCharArray();

        /* In place removal of duplicate characters*/
        while (ip_ind != arr.length)
        {
            if(arr[ip_ind] != arr[ip_ind-1])
            {
                arr[res_ind] = arr[ip_ind];
                res_ind++;
            }
            ip_ind++;

        }

        str = new String(arr);
        return str.substring(0,res_ind);
    }
```

```
        in the extra space left */
    static String removeDups(String str)
    {
        // Sort the character array
        char temp[] = str.toCharArray();
        Arrays.sort(temp);
        str = new String(temp);

        // Remove duplicates from sorted
        return removeDupsSorted(str);
    }

    // Driver Method
    public static void main(String[] args)
    {
        String str = "geeksforgeeks";
        System.out.println(removeDups(str));
    }
}
```

Python

```
# Python program to remove duplicates, the order of
# characters is not maintained in this program

# Utility function to convert string to list
def toMutable(string):
    temp = []
    for x in string:
        temp.append(x)
    return temp

# Utility function to convert string to list
def toString(List):
    return ''.join(List)

# Function to remove duplicates in a sorted array
def removeDupsSorted(List):
    res_ind = 1
    ip_ind = 1

    # In place removal of duplicate characters
    while ip_ind != len(List):
        if List[ip_ind] != List[ip_ind-1]:
            List[res_ind] = List[ip_ind]
            res_ind += 1
        ip_ind+=1

    # After above step string is efgkorskkorss.
    # Removing extra kkorss after string
    string = toString(List[0:res_ind])

    return string
```

```
# This function work in-place and fills null characters
# in the extra space left
def removeDups(string):
    # Convert string to list
    List = toMutable(string)

    # Sort the character list
    List.sort()

    # Remove duplicates from sorted
    return removeDupsSorted(List)

# Driver program to test the above functions
string = "geeksforgeeks"
print removeDups(string)

# This code is contributed by Bhavya Jain
```

C#

```
// C# program to remove duplicates, the order of
// characters is not maintained in this program
using System;

class GFG
{
    /* Method to remove duplicates in a sorted array */
    static String removeDupsSorted(String str)
    {
        int res_ind = 1, ip_ind = 1;

        // Character array for removal of duplicate characters
        char []arr = str.ToCharArray();

        /* In place removal of duplicate characters*/
        while (ip_ind != arr.Length)
        {
            if(arr[ip_ind] != arr[ip_ind-1])
            {
                arr[res_ind] = arr[ip_ind];
                res_ind++;
            }
            ip_ind++;

        }

        str = new String(arr);
        return str.Substring(0,res_ind);
    }

    /* Method removes duplicate characters from the string
    This function work in-place and fills null characters
    in the extra space left */
```

```
        // Sort the character array
        char []temp = str.ToCharArray();
        Array.Sort(temp);
        str = String.Join("",temp);

        // Remove duplicates from sorted
        return removeDupsSorted(str);
        }

        // Driver Method
        public static void Main(String[] args)
        {
            String str = "geeksforgeeks";
            Console.WriteLine(removeDups(str));
        }
    }

    // This code is contributed by 29AjayKumar
```

Output:

```
efgkors
```

Time Complexity: O(n log n) If we use some nlogn sorting algorithm instead of quicksort.

METHOD 4 (Use Hashing)

Algorithm:

```
1: Initialize:
     str  =  "test string" /* input string */
     ip_ind =  0          /* index to  keep track of location of next
                             character in input string */
     res_ind  =  0         /* index to  keep track of location of
                            next character in the resultant string */
     bin_hash[0..255] = {0,0, ….} /* Binary hash to see if character is
                                     already processed or not */
2: Do following for each character *(str + ip_ind) in input string:
              (a) if bin_hash is not set for *(str + ip_ind) then
                   // if program sees the character *(str + ip_ind) fi
                         (i)  Set bin_hash for *(str + ip_ind)
                         (ii)  Move *(str  + ip_ind) to the resultant s
                               This is done in-place.
                         (iii) res_ind++
```

```
3: Remove extra characters at the end of the resultant string.
  /*  String obtained after this step is "te sring" */
```

Implementation:

C++

```
#include <bits/stdc++.h>
using namespace std;
# define NO_OF_CHARS 256
# define bool int

/* Function removes duplicate characters from the string
This function work in-place and fills null characters
in the extra space left */
char *removeDups(char str[])
{
    bool bin_hash[NO_OF_CHARS] = {0};
    int ip_ind = 0, res_ind = 0;
    char temp;

    /* In place removal of duplicate characters*/
    while (*(str + ip_ind))
    {
        temp = *(str + ip_ind);
        if (bin_hash[temp] == 0)
        {
            bin_hash[temp] = 1;
            *(str + res_ind) = *(str + ip_ind);
            res_ind++;
        }
        ip_ind++;
    }

    /* After above step string is stringiittg.
        Removing extra iittg after string*/
    *(str+res_ind) = '\0';

    return str;
}

/* Driver code */
int main()
{
    char str[] = "geeksforgeeks";
    cout << removeDups(str);
    return 0;
}

// This code is contributed by rathbhupendra
```

C

```c
# include <stdio.h>
# include <stdlib.h>
# define NO_OF_CHARS 256
# define bool int

/* Function removes duplicate characters from the string
   This function work in-place and fills null characters
   in the extra space left */
char *removeDups(char *str)
{
  bool bin_hash[NO_OF_CHARS] = {0};
  int ip_ind = 0, res_ind = 0;
  char temp;

  /* In place removal of duplicate characters*/
  while (*(str + ip_ind))
  {
    temp = *(str + ip_ind);
    if (bin_hash[temp] == 0)
    {
        bin_hash[temp] = 1;
        *(str + res_ind) = *(str + ip_ind);
        res_ind++;
    }
    ip_ind++;
  }

  /* After above step string is stringiittg.
     Removing extra iittg after string*/
  *(str+res_ind) = '\0';

  return str;
}

/* Driver program to test removeDups */
int main()
{
    char str[] = "geeksforgeeks";
    printf("%s", removeDups(str));
    getchar();
    return 0;
}
```

Java

```java
// Java prigram to remove duplicates
import java.util.*;

class RemoveDuplicates
```

```
    void removeDuplicates(String str)
    {
        LinkedHashSet<Character> lhs = new LinkedHashSet<>();
        for(int i=0;i<str.length();i++)
            lhs.add(str.charAt(i));

        // print string after deleting duplicate elements
        for(Character ch : lhs)
            System.out.print(ch);
    }

    /* Driver program to test removeDuplicates */
    public static void main(String args[])
    {
        String str = "geeksforgeeks";
        RemoveDuplicates r = new RemoveDuplicates();
        r.removeDuplicates(str);
    }
}

// This code has been contributed by Amit Khandelwal (Amit Khandelwal 1)
```

Python

```
# Python program to remvoe duplicate characters from an
# input string
NO_OF_CHARS = 256

# Since strings in Python are immutable and cannot be changed
# This utility function will convert the string to list
def toMutable(string):
    List = []
    for i in string:
        List.append(i)
    return List

# Utility function that changes list to string
def toString(List):
    return ''.join(List)

# Function removes duplicate characters from the string
# This function work in-place and fills null characters
# in the extra space left
def removeDups(string):
    bin_hash = [0] * NO_OF_CHARS
    ip_ind = 0
    res_ind = 0
    temp = ''
    mutableString = toMutable(string)

    # In place removal of duplicate characters
    while ip_ind != len(mutableString):
        temp = mutableString[ip_ind]
```

```
                mutableString[res_ind] = mutableString[ip_ind]
                res_ind+=1
            ip_ind+=1

         # After above step string is stringiittg.
         # Removing extra iittg after string
        return toString(mutableString[0:res_ind])

# Driver program to test the above functions
string = "geeksforgeeks"
print(removeDups(string))

# A shorter version for this program is as follows
# import collections
# print ''.join(collections.OrderedDict.fromkeys(string))

# This code is contributed by Bhavya Jain
```

C#

```
// C# program to remove duplicates
using System;
using System.Collections.Generic;

class GFG
{
    /* Function removes duplicate characters
    from the string. This function work in-place */
    void removeDuplicates(String str)
    {
        HashSet<char> lhs = new HashSet<char>();
        for(int i = 0; i < str.Length; i++)
            lhs.Add(str[i]);

        // print string after deleting
        // duplicate elements
        foreach(char ch in lhs)
            Console.Write(ch);
    }

    // Driver Code
    public static void Main(String []args)
    {
        String str = "geeksforgeeks";
        GFG r = new GFG();
        r.removeDuplicates(str);
    }
}

// This code is contributed by Rajput-Ji
```

```
geksfor
```

Time Complexity: O(n)
Important Points:

- Method 1 doesn't maintain order of characters as original string, but method 2 does.
- It is assumed that number of possible characters in input string are 256. NO_OF_CHARS should be changed accordingly.
- calloc is used instead of malloc for memory allocations of counting array (count) to initialize allocated memory to '\0'. malloc() followed by memset() could also be used.
- Above algorithm also works for an integer array inputs if range of the integers in array is given. Example problem is to find maximum occurring number in an input array given that the input array contain integers only between 1000 to 1100

Method 5 (Using **IndexOf()** method) :
Prerequisite : Java **IndexOf()** method

Java

```java
// Java program too create a unique string
import java.util.*;

class IndexOf {

    // Function to make the string unique
    public static String unique(String s)
    {
        String str = new String();
        int len = s.length();

        // loop to traverse the string and
        // check for repeating chars using
        // IndexOf() method in Java
        for (int i = 0; i < len; i++)
        {
            // character at i'th index of s
            char c = s.charAt(i);

            // if c is present in str, it returns
            // the index of c, else it returns -1
            if (str.indexOf(c) < 0)
            {
                // adding c to str if -1 is returned
                str += c;
            }
```

```
        return str;
    }

    // Driver code
    public static void main(String[] args)
    {
        // Input string with repeating chars
        String s = "geeksforgeeks";

        System.out.println(unique(s));
    }
}
```

C#

```
// C# program too create a unique string
using System;

public class IndexOf
{

    // Function to make the string unique
    public static String unique(String s)
    {
        String str = "";
        int len = s.Length;

        // loop to traverse the string and
        // check for repeating chars using
        // IndexOf() method in Java
        for (int i = 0; i < len; i++)
        {
            // character at i'th index of s
            char c = s[i];

            // if c is present in str, it returns
            // the index of c, else it returns -1
            if (str.IndexOf(c) < 0)
            {
                // adding c to str if -1 is returned
                str += c;
            }
        }

        return str;
    }

    // Driver code
    public static void Main(String[] args)
    {
        // Input string with repeating chars
        String s = "geeksforgeeks";
```

```
}

// This code is contributed by Princi Singh
```

Python3

```
# Python 3 program too create a unique string

# Function to make the string unique


def unique(s):

    st = ""
    length = len(s)

    # loop to traverse the string and
    # check for repeating chars using
    # IndexOf() method in Java
    for i in range(length):

        # character at i'th index of s
        c = s[i]

        # if c is present in str, it returns
        # the index of c, else it returns - 1
        # print(st.index(c))
        if c not in st:
            # adding c to str if -1 is returned
            st += c

    return st


# Driver code
if __name__ == "__main__":

    # Input string with repeating chars
    s = "geeksforgeeks"

    print(unique(s))

    # This code is contributed by ukasp.
```

Output :

```
geksfor
```

Prerequisite : unordered_map STL C++ method

CPP

```
// C++ program too create a unique string using unordered_map

/* access time in unordered_map on is O(1) generally if no collisions occur
and therefore it helps us check if an element exists in a string in O(1)
time complexity with constant space. */

#include <bits/stdc++.h>
using namespace std;
char* removeDuplicates(char *s,int n){
  unordered_map<char,int> exists;
  int index = 0;
  for(int i=0;i<n;i++){
    if(exists[s[i]]==0)
    {
      s[index++] = s[i];
      exists[s[i]]++;
    }
  }
  return s;
}

//driver code
int main(){
  char s[] = "geeksforgeeks";
  int n = sizeof(s)/sizeof(s[0]);
  cout<<removeDuplicates(s,n)<<endl;
  return 0;
}
```

Output:

```
 geksfor
```

Time Complexity : O(n)

Auxiliary Space : O(n)

Thanks **Allen James Vinoy** for suggesting this approach.

Previous Next

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Remove all duplicates from a given string in Python**
30, Oct 17

02 **Remove three consecutive duplicates from string**
30, Aug 17

03 **Remove all consecutive duplicates from the string**
23, Oct 17

04 **Remove duplicates from a string in O(1) extra space**
26, Sep 18

05 **Remove duplicates from string keeping the order according to last occurrences**
28, Apr 20

06 **Recursively remove all adjacent duplicates**
10, Oct 13

07 **Python groupby method to remove all consecutive duplicates**
27, Dec 17

08 **Print all the duplicates in the input string**
22, Mar 09

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Rajput-Ji, princiraj1992, SHUBHAMSINGH10, amirthanand, ukasp

Article Tags : frequency-counting, Strings

Practice Tags : Strings

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org