**Remove all duplicates from a given string in Python**

def removeDuplicate(str):

    s=set(str)

    s="".join(s)

    print("Without Order:",s)

    t=""

    for i in str:

        if(i in t):

            pass

        else:

            t=t+i

        print("With Order:",t)

str="geeksforgeeks"

removeDuplicate(str)

**Reverse words in a given string**

# Python3 program to implememnt

# the above approach

# Function to reverse the words

# of a given strring

def printRev(strr):

    # Stack to store each

    # word of the strring

    strr = strr.split(" ")

    st = []

    # Store the whole strring

    # in strream

    for i in strr:

        # Push each word of the

        # into the stack

        st.append(i)

    # Print the in reverse

    # order of the words

    while len(st) > 0:

        print(st[-1], end = " ")

        del st[-1]

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

    strr = "geeks quiz practice code"

    printRev(strr)

Class Solution:

def reverseWords(self, s: str) -> str:

return ' '.join(s.split()[::-1]).strip()

**Run Length Encoding in Python**

For example, if the input string is ‘wwwwaaadexxxxxx’, then the function should return ‘w4a3d1e1x6’.

Examples:

Input : str = 'wwwwaaadexxxxxx'

Output : 'w4a3d1e1x6'

|  |
| --- |
| # Python code for run length encoding  from collections import OrderedDict  def runLengthEncoding(input):        # Generate ordered dictionary of all lower      # case alphabets, its output will be      # dict = {'w':0, 'a':0, 'd':0, 'e':0, 'x':0}      dict=OrderedDict.fromkeys(input, 0)        # Now iterate through input string to calculate      # frequency of each character, its output will be      # dict = {'w':4,'a':3,'d':1,'e':1,'x':6}      for ch in input:          dict[ch] += 1        # now iterate through dictionary to make      # output string from (key,value) pairs      output = ''      for key,value in dict.items():           output = output + key + str(value)      return output    # Driver function  if \_\_name\_\_ == "\_\_main\_\_":      input="wwwwaaadexxxxxx"      print (runLengthEncoding(input)) |

Output:

'w4a3d1e1x6'

def encode(message):

    encoded\_message = ""

    i = 0

    while (i <= len(message)-1):

        count = 1

        ch = message[i]

        j = i

        while (j < len(message)-1):

            if (message[j] == message[j+1]):

                count = count+1

                j = j+1

            else:

                break

        encoded\_message=encoded\_message+str(count)+ch

        i = j+1

    return encoded\_message

#Provide different values for message and test your program

encoded\_message=encode("ABBBBCCCCCCCCAB")

print(encoded\_message)

Remove all adjacent duplicate characters in a string python

class Solution:

def removeDuplicates(self, S: str) -> str:

s = ""

for letter in S:

if len(s) == 0 or letter != s[-1]:

s += letter

else:

s = s[:-1]

return s;

#TC: O(N2)

#SC: O(N)

#initiate with empty output stack

output = []

for ch in S:

if output and ch == output[-1]:

output.pop()

else:

output.append(ch)

return ''.join(output)

#TC: O(N)

#SC: O(N-D)

**Given a string, find its first non-repeating character**

Given a string, find the first non-repeating character in it. For example, if the input string is “GeeksforGeeks”, then the output should be ‘f’ and if the input string is “GeeksQuiz”, then the output should be ‘G’.s

def firstNonRepeatingChar(str1):

   char\_order = []

   counts = {}

   for c in str1:

      if c in counts:

         counts[c] += 1

      else:

         counts[c] = 1

         char\_order.append(c)

   for c in char\_order:

      if counts[c] == 1:

      return c

   return None

print(firstNonRepeatingChar('PythonforallPythonMustforall'))

print(firstNonRepeatingChar('tutorialspointfordeveloper'))

print(firstNonRepeatingChar('AABBCC'))

**Result**

M

u

None

Find the smallest window in a string containing all characters of another string

**49. Group Anagrams**

Medium

4760217Add to ListShare

Given an array of strings strs, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Example 1:**

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

**Example 2:**

**Input:** strs = [""]

**Output:** [[""]]

s

class Solution:

def groupAnagrams(self, strs: List[str]) -> List[List[str]]:

if not strs:

return []

d = {}

for word in strs:

key = "".join(sorted(word))

if key not in d:

d[key] = [word]

else:

d[key].append(word)

return d.values()

**Rearrange characters to form palindrome if possible**

Given a string, convert the string to palindrome without any modifications like adding a character, removing a character, replacing a character etc.  
Examples:

Input : "mdaam"

Output : "madam" or "amdma"

Input : "abb"

Output : "bab"

Input : "geeksforgeeks"

Output : "No Palindrome"

Imp Point:- All the charaters should present even number of times except 1.

1. Count occurrences of all characters.  
   2. Count odd occurrences. If this count is greater than 1 or is equal to 1 and length of the string is even then obviously palindrome cannot be formed from the given string.

3. Initialize two empty strings firstHalf and secondHalf.  
4. Traverse the map. For every character with count as count, attach count/2 characters to end of firstHalf and beginning of secondHalf.  
5. Finally return the result by appending firstHalf and secondHalf

def canFormPalindrome(s):

count=[0]\*256

odd=0

for i in range(len(s)):

count[ord(s[i])]+=1

for i in count:

if i%2!=0:

odd+=1

if(odd>1):

return False

return True

if \_\_name\_\_=='\_\_main\_\_':

s="madmac"

val=canFormPalindrome(s)

if val==True:

print("Palindrome can be formed")

else:

print("Palindrome cannot be formed")

**Reorder Data in Log Files**

You have an array of logs.  Each log is a space delimited string of words.

For each log, the first word in each log is an alphanumeric *identifier*.  Then, either:

* Each word after the identifier will consist only of lowercase letters, or;
* Each word after the identifier will consist only of digits.

We will call these two varieties of logs *letter-logs* and *digit-logs*.  It is guaranteed that each log has at least one word after its identifier.

Reorder the logs so that all of the letter-logs come before any digit-log.  The letter-logs are ordered lexicographically ignoring identifier, with the identifier used in case of ties.  The digit-logs should be put in their original order.

Return the final order of the logs.

**Example 1:**

**Input:** logs = ["dig1 8 1 5 1","let1 art can","dig2 3 6","let2 own kit dig","let3 art zero"]

**Output:** ["let1 art can","let3 art zero","let2 own kit dig","dig1 8 1 5 1","dig2 3 6"]

letters, digits = [], []

for val in logs:

if val[-1].isalpha():

arr = val.split(" ")

arr = arr[1:] + arr[:1]

letters.append(" ".join(arr))

else:

digits.append(val)

letters = sorted(letters)

for i, letter in enumerate(letters):

arr = letter.split(" ")

arr = arr[-1:] + arr[:-1]

letters[i] = " ".join(arr)

return letters + digits

def reorderLogFiles(self, logs: List[str]) -> List[str]:

letter =[]

digit =[]

result = []

for log in logs:

if (log.split()[1]).isdigit():

digit.append(log)

else:

letter.append(log.split())

letter.sort(key = lambda x :x[0])

letter.sort(key = lambda x :x[1:])

for i in range(len(letter)):

result.append(" ".join(letter[i]))

result.extend(digit)

return result

**Sort Characters By Frequency**

Given a string, sort it in decreasing order based on the frequency of characters.

**Example 1:**

**Input:**

"tree"

**Output:**

"eert"

**Explanation:**

'e' appears twice while 'r' and 't' both appear once.

So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.

**Example 2:**

**Input:**

"cccaaa"

**Output:**

"cccaaa"

**Explanation:**

Both 'c' and 'a' appear three times, so "aaaccc" is also a valid answer.

Note that "cacaca" is incorrect, as the same characters must be together.

# Python3 implementation to Sort strings

# according to the frequency of

# characters in ascending order

# Returns count of character in the string

def countFrequency(string ,  ch) :

    count = 0;

    for i in range(len(string)) :

        # Check for vowel

        if (string[i] == ch) :

            count += 1;

    return count;

# Function to sort the string

# according to the frequency

def sortArr(string) :

    n = len(string);

    # Vector to store the frequency of

    # characters with respective character

    vp = [];

    # Inserting frequency

    # with respective character

    # in the vector pair

    for i in range(n) :

        vp.append((countFrequency(string, string[i]), string[i]));

    # Sort the vector, this will sort the pair

    # according to the number of characters

    vp.sort();

    # Print the sorted vector content

    for i in range(len(vp)) :

        print(vp[i][1],end="");

# Driver code

if \_\_name\_\_ == "\_\_main\_\_" :

    string = "geeksforgeeks";

    sortArr(string);

    # This code is contributed by Yash\_R

class Solution(object):

def frequencySort(self, s):

"""

:type s: str

:rtype: str

"""

dic = {}

for c in s:

if c not in dic: dic[c] = 1

else: dic[c] += 1

sorted\_keys = [x for x,\_ in sorted(zip(dic.keys(),dic.values()), key= lambda y: y[1], reverse = True)]

return ''.join([c\*dic[c] for c in sorted\_keys])

**Reorganize String**

Given a string S, check if the letters can be rearranged so that two characters that are adjacent to each other are not the same.

If possible, output any possible result.  If not possible, return the empty string.

**Example 1:**

**Input:** S = "aab"

**Output:** "aba"

**Example 2:**

**Input:** S = "aaab"

**Output:** ""

# Python3 implementation of the approach

# Function that returns true if it is possible

# to rearrange the characters of the String

# such that no two consecutive characters are same

def isPossible(Str):

    # To store the frequency of

    # each of the character

    freq = dict()

    # To store the maximum frequency so far

    max\_freq = 0

    for j in range(len(Str)):

        freq[Str[j]] = freq.get(Str[j], 0) + 1

        if (freq[Str[j]] > max\_freq):

            max\_freq = freq[Str[j]]

    # If possible

    if (max\_freq <= (len(Str) - max\_freq + 1)):

        return True

    return False

# Driver code

Str = "geeksforgeeks"

if (isPossible(Str)):

    print("Yes")

else:

    print("No")

# This code is contributed by Mohit Kumar

class Solution:

def reorganizeString(self, S: str) -> str:

counts = defaultdict(int)

maxc = 0

for c in S:

counts[c] += 1

maxc = max(maxc, counts[c])

if maxc > math.ceil(len(S)/2):

return ""

heap = MaxHeap([(c,s) for s,c in counts.items()])

res = ""

while heap:

nxt = heap.pop()

if res and res[-1] == nxt[1]:

tmp = heap.pop()

heap.push(nxt)

nxt = tmp

res += nxt[1]

if nxt[0] > 1:

heap.push((nxt[0]-1, nxt[1]))

return res