Rearrange characters in a string such that no two adjacent are same

Difficulty Level : Hard • Last Updated : 20 Jun, 2022

Given a **string** with repeated characters, the task is to rearrange characters in a string so that no two adjacent characters are same.
Note : It may be assumed that the string has only lowercase English alphabets.

Examples:

> *Input: aaabc*
> *Output: abaca*
>
> *Input: aaabb*
> *Output: ababa*
>
> *Input: aa*
> *Output: Not Possible*
>
> *Input: aaaabc*
> *Output: Not Possible*

Recommended PracticeRearrange charactersTry It!

Approach (using priority_queue):
The idea is to put the highest frequency character first (a greedy approach). We use a priority queue (Or Binary Max Heap) and put all characters and ordered by their frequencies (highest frequency character at root). We one by one take the highest frequency character from the heap and add it to result. After we add, we decrease the frequency of the character and we temporarily move this character out of priority queue so that it is not picked next time.

We have to follow the step to solve this problem, they are:

- Build a Priority_queue or max_heap, **pq** that stores characters and their frequencies.
  - Priority_queue or max_heap is built on the bases of the frequency of character.
- Create a temporary Key that will be used as the previously visited element (the previous element in the resultant string. Initialize it { char = '#' , freq = '-1' }
- While **pq** is not empty.
  - Pop an element and add it to the result.
  - Decrease frequency of the popped element by '1'
  - Push the previous element back into the priority_queue if it's frequency > '0'
  - Make the current element as the previous element for the next iteration.
- If the length of the resultant string and original string is not equal,

C++

```cpp
// C++ program to rearrange characters in a string
// so that no two adjacent characters are same.
#include <bits/stdc++.h>
using namespace std;

const int MAX_CHAR = 26;

struct Key {
    int freq; // store frequency of character
    char ch;

    // function for priority_queue to store Key
    // according to freq
    bool operator<(const Key& k) const
    {
        return freq < k.freq;
    }
};

// Function to rearrange character of a string
// so that no char repeat twice
void rearrangeString(string str)
{
    int n = str.length();

    // Store frequencies of all characters in string
    int count[MAX_CHAR] = { 0 };
    for (int i = 0; i < n; i++)
        count[str[i] - 'a']++;

    // Insert all characters with their frequencies
    // into a priority_queue
    priority_queue<Key> pq;
    for (char c = 'a'; c <= 'z'; c++) {
          int val = c - 'a';
        if (count[val]) {
            pq.push(Key{ count[val], c });
    // work as the previous visited element
    // initial previous element be. ( '#' and
    // it's frequency '-1' )
    Key prev{ -1, '#' };

    // traverse queue
    while (!pq.empty()) {
        // pop top element from queue and add it
        // to string.
        Key k = pq.top();
        pq.pop();
        str = str + k.ch;

        // IF frequency of previous character is less
        // than zero that means it is useless, we
        // need not to push it
        if (prev.freq > 0)
            pq.push(prev);

        // make current character as the previous 'char'
        // decrease frequency by 'one'
        (k.freq)--;
        prev = k;
    }

    // If length of the resultant string and original
    // string is not same then string is not valid
    if (n != str.length())
        cout << " Not valid String " << endl;

    else // valid string
        cout << str << endl;
}

// Driver program to test above function
int main()
{
    string str = "bbbaa";
    rearrangeString(str);
    return 0;
```

Java

```
// Java program to rearrange characters in a string
// so that no two adjacent characters are same.
import java.io.*;
import java.util.*;

class KeyComparator implements Comparator<Key> {

    // Overriding compare()method of Comparator
    public int compare(Key k1, Key k2)
    {
        if (k1.freq < k2.freq)
            return 1;
        else if (k1.freq > k2.freq)
            return -1;
        return 0;
    }
}

class Key {
    int freq; // store frequency of character
    char ch;
    Key(int val, char c)
    {
        freq = val;
        ch = c;
    }
}

class GFG {
    static int MAX_CHAR = 26;

    // Function to rearrange character of a string
    // so that no char repeat twice
    static void rearrangeString(String str)
    {
        int n = str.length();
```

```
// Insert all characters with their frequencies
// into a priority_queue
PriorityQueue<Key> pq
    = new PriorityQueue<>(new KeyComparator());
for (char c = 'a'; c <= 'z'; c++) {
    int val = c - 'a';
    if (count[val] > 0)
        pq.add(new Key(count[val], c));
}

// 'str' that will store resultant value
str = "";

// work as the previous visited element
// initial previous element be. ( '#' and
// it's frequency '-1' )
Key prev = new Key(-1, '#');

// traverse queue
while (pq.size() != 0) {

    // pop top element from queue and add it
    // to string.
    Key k = pq.peek();
    pq.poll();
    str = str + k.ch;

    // If frequency of previous character is less
    // than zero that means it is useless, we
    // need not to push it
    if (prev.freq > 0)
        pq.add(prev);

    // make current character as the previous 'char'
    // decrease frequency by 'one'
    (k.freq)--;
    prev = k;
}

// If length of the resultant string and original
```

```
            System.out.println(str);
    }

    // Driver program to test above function
    public static void main(String args[])
    {
        String str = "bbbaa";
        rearrangeString(str);
    }
}

// This code is contributed by rachana soma
```

Python3

```
# Python program to rearrange characters in a string
# so that no two adjacent characters are same.
from heapq import heappush, heappop
from collections import Counter

# A key class for readability
class Key:
    def __init__(self, character: str, freq: int) -> None:
        self.character = character
        self.freq = freq

    def __lt__(self, other: "Key") -> bool:
        return self.freq > other.freq


# Function to rearrange character of a string
# so that no char repeat twice
def rearrangeString(str: str):
    n = len(str)
    # creating a frequency hashmap
    count = dict()
    for i in str:
        count[ord(i)] = count.get(ord(i), 0) + 1

    pq = []
    # null character for default previous checking
    prev = Key('#', -1)
    str = ""

    while pq:
        key = heappop(pq)
        str += key.character

        # Since one character is already added
        key.freq -= 1

        # We avoid inserting if the frequency drops to 0
        if prev.freq > 0:
            heappush(pq, prev)

        prev = key

    if len(str) != n:
        print("Not a Valid str")
    else:
        print(str)


# Driver Code
if __name__ == "__main__":
    string = "bbbaa"
    rearrangeString(string)

    # This code is contributed by kraanzu.
```

Output

```
babab
```

Time complexity : O(n log(n))

Here n is length of the string

Auxiliary Space: O(n)

Another approach: Another approach is to fill all the even positions of the result string first, with the highest frequency character. If there are still some even positions remaining, fill them first. Once even positions are done, then fill the odd positions. This way, we can ensure that no two adjacent characters are the same.

C++14

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

char getMaxCountChar(const vector<int>& count)
{
    int max = 0;
    char ch;
    for (int i = 0; i < 26; i++) {
        if (count[i] > max) {
            max = count[i];
            ch = 'a' + i;
        }
    }

    return ch;
}

string rearrangeString(string S)
{

    int n = S.size();
    if (!n)
        return "";

    vector<int> count(26, 0);
    for (auto ch : S)
        count[ch - 'a']++;

    char ch_max = getMaxCountChar(count);
    int maxCount = count[ch_max - 'a'];
    string res(n, ' ');

    int ind = 0;
    // filling the most frequently occurring char in the even
    // indices
    while (maxCount) {
        res[ind] = ch_max;
        ind = ind + 2;
        maxCount--;
    }
    count[ch_max - 'a'] = 0;

    // now filling the other Chars, first filling the even
    // positions and then the odd positions
    for (int i = 0; i < 26; i++) {
        while (count[i] > 0) {
            ind = (ind >= n) ? 1 : ind;
            res[ind] = 'a' + i;
            ind += 2;
            count[i]--;
        }
    }
    return res;
}

// Driver program to test above function
int main()
{
    string str = "bbbaa";
    string res = rearrangeString(str);
    if (res == "")
        cout << "Not valid string" << endl;
    else
        cout << res << endl;
    return 0;
}
```

Java

```
class GFG {
  static char getMaxCountChar(int[] count)
  {
    int max = 0;
    char ch = 0;
    for (int i = 0; i < 26; i++) {
      if (count[i] > max) {
        max = count[i];
        ch = (char)((int)'a' + i);
      }
    }
    return ch;
  }

  static String rearrangeString(String S)
  {

    int n = S.length();
    if (n==0)
      return "";

    int[]count = new int[26];
    for(int i=0;i<26;i++){
      count[i] = 0;
    }
    for (char ch : S.toCharArray()){
      count[(int)ch - (int)'a']++;
    }


    char ch_max = getMaxCountChar(count);
    int maxCount = count[(int)ch_max - (int)'a'];

    // check if the result is possible or not
    if (maxCount > (n + 1) / 2)
      return "";

    String res = "";
    for(int i=0;i<n;i++){
      res += ' ';
      // indices
      while (maxCount > 0) {
        res = res.substring(0,ind) + ch_max + res.substring(ind+1);
        ind = ind + 2;
        maxCount--;
      }
      count[(int)ch_max - (int)'a'] = 0;

      // now filling the other Chars, first filling the even
      // positions and then the odd positions
      for (int i = 0; i < 26; i++) {
        while (count[i] > 0) {
          ind = (ind >= n) ? 1 : ind;
          res = res.substring(0,ind) + (char)((int)'a' + i) + res.substring
          ind += 2;
          count[i]--;
        }
      }
      return res;
    }

    // Driver Code
    public static void main(String args[])
    {
      String str = "bbbaa";
      String res = rearrangeString(str);
      if (res == "")
        System.out.println("Not valid string");
      else
        System.out.println(res);
    }
  }

  // This code is contributed by shinjanpatra
```

Python3

```
# Python program for rearranging characters in a string such
# that no two adjacent are same
```

```
        if count[i] > maxCount:
            maxCount = count[i]
            maxChar = chr(i + ord('a'))

    return maxCount, maxChar

# Main function for rearranging the characters
def rearrangeString(S):
    n = len(S)

    # if length of string is None return False
    if not n:
        return False

    # create a hashmap for the alphabets
    count = [0] * 26
    for char in S:
        count[ord(char) - ord('a')] += 1


    maxCount, maxChar = getMaxCountChar(count)

    # if the char with maximum frequency is more than the half of the
    # total length of the string than return False
    if maxCount > (n + 1) // 2:
        return False

    # create a list for storing the result
    res = [None] * n

    ind = 0

    # place all occurrences of the char with maximum frequency in
    # even positions
    while maxCount:
        res[ind] = maxChar
        ind += 2
        maxCount -= 1

    # replace the count of the char with maximum frequency to zero
    # as all the maxChar are already placed in the result
```

```
  for i in range(26):
      while count[i] > 0:
          if ind >= n:
              ind = 1
          res[ind] = chr(i + ord('a') )
          ind += 2
          count[i] -= 1


  # convert the result list to string and return
  return ''.join(res)

# Driver Code
str = 'bbbaa'
res = rearrangeString(str)
if res:
  print(res)
else:
  print('Not valid string')

# This code is contributed by Manish Thapa
```

Javascript

```
<script>

// JavaScript program for rearranging characters in a string such
// that no two adjacent are same

// Function to find the char with maximum frequency in the given
// string
function getMaxCountChar(count){
let maxCount = 0
let maxChar
for(let i = 0; i < 26; i++){
    if(count[i] > maxCount){
        maxCount = count[i]
        maxChar = String.fromCharCode(i + ('a').charCodeAt(0))
  }
```

```
// Main function for rearranging the characters
function rearrangeString(S){
let n = S.length

// if length of string is None return false
if(!n)
    return false

// create a hashmap for the alphabets
let count = new Array(26).fill(0)
for(let char of S)
    count[char.charCodeAt(0) - ('a').charCodeAt(0)] += 1

let [maxCount, maxChar] = getMaxCountChar(count)

// if the char with maximum frequency is more than the half of the
// total length of the string than return false
if(maxCount > Math.floor((n + 1) / 2))
    return false

// create a list for storing the result
let res = new Array(n)

let ind = 0

// place all occurrences of the char with maximum frequency in
// even positions
while(maxCount){
    res[ind] = maxChar
    ind += 2
    maxCount -= 1
}

// replace the count of the char with maximum frequency to zero
// as all the maxChar are already placed in the result
count[maxChar.charCodeAt(0) - 'a'.charCodeAt(0)] = 0

// place all other char in the result starting from remaining even
// positions and then place in the odd positions
for(let i = 0; i < 26; i++)
{
        res[ind] = String.fromCharCode(i + ('a').charCodeAt(0))
        ind += 2
        count[i] -= 1
  }
}

// convert the result list to string and return
return res.join('')
}

// Driver Code
let str = 'bbbaa'
let res = rearrangeString(str)
if(res)
  document.write(res,"</br>")
else
  document.write('Not valid string',"</br>")

// This code is contributed by shinjanpatra

</script>
```

Output

```
babab
```

Time complexity : O(n)
Auxiliary Space : O(n+26) where 26 is the size of the vocabulary.

This article is contributed by **Nishant Singh** . If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Like 165

Previous

Removing punctuations from a given string

Next

Program to check if input is an integer or a string

RECOMMENDED ARTICLES

Page : 1 2 3

25, Jul 19

04 Rearrange string such that no pair of adjacent characters are of the same type
21, Sep 20

08 String with k distinct characters and no same characters adjacent
25, Oct 17

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Easy Normal Medium Hard Expert

Improved By : rachana soma, gp6, ankit_bansal_2020, Anshul Sharma 2, ArjunSharma, yezhengli9, SwathiKedarasetty, iammanish041, karthikeshwar, kunalgupta20, surindertarika1234, simranarora5sos, shinjanpatra, RishabhPrabhu, kraanzu, abhijeet19403

Article Tags : Amazon, STL, Greedy, Heap, Strings

Practice Tags : Amazon, Strings, Greedy, Heap, STL

Improve Article Report Issue

NodeJS