Rearrange a string so that all same characters become d distance away

Given a string and a positive integer d. Some characters may be repeated in the given string. Rearrange characters of the given string such that the same characters become d distance away from each other. Note that there can be many possible rearrangements, the output should be one of the possible rearrangements. If no such arrangement is possible, that should also be reported. Expected time complexity is O(n) where n is length of input string.

```
Examples:
Input:  "abb", d = 2
Output: "bab"

Input:  "aacbbc", d = 3
Output: "abcabc"

Input: "geeksforgeeks", d = 3
Output: egkegkesfesor

Input:  "aaa",  d = 2
Output: Cannot be rearranged
```

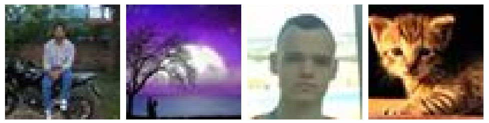
We strongly recommend to minimize the browser and try this yourself first.
Hint: Alphabet size may be assumed as constant (256) and extra space may be used.

Solution: The idea is to count frequencies of all characters and consider the most frequent character first and place all occurrences of it as close as possible. After the most frequent character is placed, repeat the same process for remaining characters.

Google™ Custom Search

GeeksforGeeks

52,731 people like GeeksforGeeks.



Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion
Geometric Algorithms

1) Let the given string be str and size of string be n

2) Traverse str, store all characters and their frequencies in a Max Heap MH. The value of frequency decides the order in MH, i.e., the most frequent character is at the root of MH.

3) Make all characters of str as '\0'.

4) Do following while MH is not empty.
…**a)** Extract the Most frequent character. Let the extracted character be x and its frequency be f.
…**b)** Find the first available position in str, i.e., find the first '\0' in str.
…**c)** Let the first position be p. Fill x at p, p+d,.. p+(f-1)d

Following is C++ implementation of above algorithm.

```
// Rearrange a string so that all same characters become at least d
// distance away
#include <iostream>
#include <cstring>
#include <cstdlib>
#define MAX 256
using namespace std;

// A structure to store a character 'c' and its frequency 'f'
// in input string
struct charFreq {
    char c;
    int f;
};

// A utility function to swap two charFreq items.
void swap(charFreq *x, charFreq *y) {
    charFreq z = *x;
    *x = *y;
    *y = z;
}

// A utility function to maxheapify the node freq[i] of a heap
// stored in freq[]
void maxHeapify(charFreq freq[], int i, int heap_size)
{
    int l = i*2 + 1;
    int r = i*2 + 2;
    int largest = i;
```

Popular Posts

All permutations of a given string
Memory Layout of C Programs
Understanding "extern" keyword in C
Median of two sorted arrays
Tree traversal without recursion and without stack!
Structure Member Alignment, Padding and Data Packing
Intersection point of two Linked Lists
Lowest Common Ancestor in a BST.
Check if a binary tree is BST or not
Sorted Linked List to Balanced BST

```
    if (l < heap_size && freq[l].f > freq[i].f)
        largest = l;
    if (r < heap_size && freq[r].f > freq[largest].f)
        largest = r;
    if (largest != i)
    {
        swap(&freq[i], &freq[largest]);
        maxHeapify(freq, largest, heap_size);
    }
}

// A utility function to convert the array freq[] to a max heap
void buildHeap(charFreq freq[], int n)
{
    int i = (n - 1)/2;
    while (i >= 0)
    {
        maxHeapify(freq, i, n);
        i--;
    }
}

// A utility function to remove the max item or root from max heap
charFreq extractMax(charFreq freq[], int heap_size)
{
    charFreq root = freq[0];
    if (heap_size > 1)
    {
        freq[0] = freq[heap_size-1];
        maxHeapify(freq, 0, heap_size-1);
    }
    return root;
}

// The main function that rearranges input string 'str' such that
// two same characters become d distance away
void rearrange(char str[], int d)
{
    // Find length of input string
    int n = strlen(str);

    // Create an array to store all characters and their
    // frequencies in str[]
    charFreq freq[MAX] = {{0, 0}};

    int m = 0; // To store count of distinct characters in str[]
    // Traverse the input string and store frequencies of all
    // characters in freq[] array.
    for (int i = 0; i < n; i++)
    {
        char x = str[i];

        // If this character has occurred first time, increment m
        if (freq[x].c == 0)
            freq[x].c = x, m++;

        (freq[x].f)++;
        str[i] = '\0';  // This change is used later
    }

    // Build a max heap of all characters
    buildHeap(freq, MAX);

    // Now one by one extract all distinct characters from max heap
    // and put them back in str[] with the d distance constraint
    for (int i = 0; i < m; i++)
    {
        charFreq x = extractMax(freq, MAX-i);

        // Find the first available position in str[]
        int p = i;
        while (str[p] != '\0')
            p++;

        // Fill x.c at p, p+d, p+2d, .. p+(f-1)d
        for (int k = 0; k < x.f; k++)
        {
            // If the index goes beyond size, then string cannot
            // be rearranged.
            if (p + d*k >= n)
            {
                cout << "Cannot be rearranged";
                exit(0);
            }
            str[p + d*k] = x.c;
        }
    }
}

// Driver program to test above functions
int main()
{
    char str[] = "aabbcc";
```

Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 13 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 33 minutes ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 33 minutes ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 2 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 2 hours ago

```
    rearrange(str, 3);
    cout << str;
}
```

Output:

```
abcabc
```

Algorithmic Paradigm: Greedy Algorithm

Time Complexity: Time complexity of above implementation is O(n + mLog(MAX)). Here n is the length of str, m is count of distinct characters in str[] and MAX is maximum possible different characters. MAX is typically 256 (a constant) and m is smaller than MAX. So the time complexity can be considered as O(n).

More Analysis:
The above code can be optimized to store only m characters in heap, we have kept it this way to keep the code simple. So the time complexity can be improved to O(n + mLogm). It doesn't much matter through as MAX is a constant.

Also, the above algorithm can be implemented using a O(mLogm) sorting algorithm. The first steps of above algorithm remain same. Instead of building a heap, we can sort the freq[] array in non-increasing order of frequencies and then consider all characters one by one from sorted array.

We will soon be covering an extended version where same characters should be moved at least d distance away.

This article is contributed by **Himanshu Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Tpoics:

- Printing Longest Common Subsequence
- Suffix Array | Set 2 (nLogn Algorithm)
- Recursively remove all adjacent duplicates
- Find the first non-repeating character from a stream of characters
- Dynamic Programming | Set 33 (Find if a string is interleaved of two other strings)
- Remove “b” and “ac” from a given string
- Dynamic Programming | Set 29 (Longest Common Substring)
- Write your own atoi()

8 Tweet 3 1

Writing code in comment? Please use **ideone.com** and share the link here.

13 Comments GeeksforGeeks

Sort by Newest

Join the discussion

Join the discussion…

codex • 22 days ago

can we implement it without using heap????

Reply • Share ›

jitender • a month ago

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

struct node // structure to store character and it's frequency
{
char c;
int count;
};

void sameCharacter_K_Distance(char *str , int d) // function for rearrange the
{

int size=strlen(str); // length of string

struct node* arr=(struct node*)malloc(sizeof(struct node)*size); // struct node

int i,j,n=0,l=0,k=0;

/* code for calculate each character and it's frequency */
```

see more

Reply • Share ›

bharat • 2 months ago

if string is "AAABBBCCC" and d = 2.....

then can we arrage like this..... "ABACACBCB" (taking a relative postion from suggesttions please??

Reply • Share ›

GeeksforGeeks Mod • 2 months ago

All, thanks for your inputs. The above provided solution works only for the prob
moved exact d distance away. We have updated the problem statement. We
version where same characters should be moved at least d distance away.

Reply • Share ›

Guest • 2 months ago

There is logical error in for loop, when you are re-arranging chars. Correct loo

for (int i = 0; i < m; i++)

{

charFreq x = extractMax(freq, MAX-i);

// Find the first available position in str[]

int p = i;

while (str[p] != '\0')

p++;

// Fill x.c at p, p+d, p+2d, .. p+(f-1)d

for (int k = 0; k < x.f; k++)

{

see more

Reply • Share ›

GeeksforGeeks Mod Guest • 2 months ago

Guest, this fix doesn't work for the example provided by uuuouou in bel

Reply • Share ›

RajKumar Rampalli • 2 months ago

Can we implement it using below simple logic.

1. Traverse string and note down the all occurrences of characters in 2-D arra

Ex: aaabbbccc becomes a|3|b|3|c|3 and acbcaa becomes a|3|c|2|b|1

2. Now, take each character from each row of 2-D array and make up the new starting from index 0

Ex: i) a|3|b|3|c|3

In 1st iteration --> abc

In 2nd iteration --> abcabc

In 3rd iteration ---> abcabcabc

ii) a|3|c|2|b|1

In 1st iteration --> acb

In 2nd iteration --> acbac

In 3rd iteration ---> acbaca

Suggestions are welcome.!

Reply • Share ›

Kartik RajKumar Rampalli • 2 months ago

This doesn't seem to work for

aaabbcdefg

d = 3

The 2-D array is

a3b2c1d1e1f1g1

After first pass

abcdefg

After second pass
abcdefgab

3rd pass fails to place a and b

But string can be rearranged abcabdeafg.

I think both the approaches should be combined.

Let me know your thoughts.

• Reply • Share ›

RajKumar Rampalli Kartik • 2 months ago
Karthik, I think the proposed Himanshu Gupta's logic will work f
leading everyone in wrong path. Since d=2 --> a--a--a --> ab-ak

• Reply • Share ›

uuuouou • 2 months ago
I'm sorry, but your algorithm will fail when the string is "aaabbbccc" and d = 2.
algorithm:
(1)a_a_a____
(2)ababab___
(3)can not be rearranged
But in fact we can arrange the string into "abcabcabc" to meet the requiremen

• Reply • Share ›

Toney uuuouou • 2 months ago
No, in your answer 'abcabcabc', same characters are not exactly 2 ch

• Reply • Share ›

flyingbird Toney • a month ago
if >= distance, I think this way does not work. we need to consi
lowered after it is put into the spot. Cannot allocate most freque

advance. The check should be done for each spot.

Reply • Share ›

GeeksforGeeks Mod uuuouou • 2 months ago

Thanks for pointing this out. We will soon update this post with correct

Reply • Share ›

Subscribe Add Disqus to your site