【华为OD机考 统一考试机试C卷】连续字母长度（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

题目描述

给定一个字符串，只包含大写字母，求在包含同一字母的子串中，长度第 k 长的子串的长度，相同字母只取最长的那个子串。

输入描述

第一行有一个子串(1<长度<=100)，只包含大写字母。

第二行为 k的值

输出描述

输出连续出现次数**第k多**的字母的次数。

用例1

输入

```
AAAAHHHBBCDHHHH

```

```
3
```

输出

```
2
```

说明

> 同一字母连续出现的最多的是A和H，四次；
> 第二多的是H，3次，但是H已经存在4个连续的，故不考虑；
> 下个最长子串是BB，所以最终答案应该输出2。

用例2

输入

```
AABAAA
2
```

输出

```
1
```

说明

同一字母连续出现的最多的是A，三次；
第二多的还是A，两次，但A已经存在最大连续次数三次，故不考虑；
下个最长子串是B，所以输出1。

用例3

输入

```
ABC
4
```

输出

```
-1
```

说明
只含有3个包含同一字母的子串，小于k，输出-1

用例4

输入

```
ABC
2
```

输出

```
1
```

说明
三个子串长度均为1，所以此时k = 1，k=2，k=3这三种情况均输出1。特此说明，避免歧义。

C++

```
#include <iostream>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <regex>
using namespace std;


int main() {
  string str;
  int k = 0;
  cin >> str;
  cin >> k;
  unordered_set<char> charSet;
  for (char c : str) {
```

```
    charSet.insert(c);
  }
  unordered_map<char, int> charMap;
  for (char c : charSet) {
    const regex reg(string(1, c) + "+");
    sregex_iterator it(str.begin(), str.end(), reg);
    while (it != sregex_iterator()) {
      int repeatTimes = it->str().length();
      charMap[c] = charMap.count(c) ? max(charMap[c], repeatTimes) : repeatTimes;
      ++it;
    }
  }
  vector<int> values;
  for (auto it : charMap) {
    values.push_back(it.second);
  }
  sort(values.begin(), values.end(), greater<int>());
  int rt = k > values.size() ? -1 : values[k - 1];
  cout << rt << endl;
  return 0;
}
```

java

```
import java.util.Scanner;
import java.util.HashSet;
import java.util.HashMap;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.ArrayList;
import java.util.Collections;

public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String str = input.next();
        int k = input.nextInt();
        HashSet<Character> charSet = new HashSet<>();
        for (char c : str.toCharArray()) {
            charSet.add(c);
```

```
        }
        HashMap<Character, Integer> charMap = new HashMap<>();
        for (char c : charSet) {
            Pattern pattern = Pattern.compile(String.valueOf(c) + "+");
            Matcher matcher = pattern.matcher(str);
            while (matcher.find()) {
                int repeatTimes = matcher.group().length();
                if (charMap.containsKey(c)) {
                    charMap.put(c, Math.max(charMap.get(c), repeatTimes));
                } else {
                    charMap.put(c, repeatTimes);
                }
            }
        }
        ArrayList<Integer> values = new ArrayList<>(charMap.values());
        Collections.sort(values, Collections.reverseOrder());
        int rt = k > values.size() ? -1 : values.get(k - 1);
        System.out.println(rt);
    }
}
```

javaScript

```

const { listeners } = require("process");
const readline = require("readline");
const { isNumber } = require("util");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (str) => {
 rl.on("line", (k) => {

 let set = new Set(str);
  let obj = {};
  for (let letter of set) {
    const reg = new RegExp(`${letter}+`, "g");
```

```
    while (true) {
      let res = reg.exec(str);
      if (res === null) {
        break;
      } else {
        let repeatTimes = res[0].length;
        obj[letter] = obj[letter]
          ? Math.max(obj[letter], repeatTimes)
          : repeatTimes;
      }
    }
  }
let res=Object.values(obj).sort((a, b) => b - a)[k - 1] ?? -1;
    console.log( res )

});


});
```

python

```
import re

str = input()
k = int(input())
charSet = set(str)
charMap = {}
for c in charSet:
    reg = re.compile(c + "+")
    it = re.finditer(reg, str)
    for match in it:
        repeatTimes = len(match.group())
        if c in charMap:
            charMap[c] = max(charMap[c], repeatTimes)
        else:
            charMap[c] = repeatTimes
```

```
values = list(charMap.values())
values.sort(reverse=True)
rt = -1 if k > len(values) else values[k-1]
print(rt)
```

C语言

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

int main() {
    char str[101]; // 存储输入的字符串，最大长度100
    int k, charMaxCount[26] = {0}, count = 0, maxCount = 0;
    scanf("%s", str); // 读取字符串
    scanf("%d", &k); // 读取k的值

    // 计算每个字符的最大连续出现次数
    int len = strlen(str);
    for (int i = 0; i < len; i++) {
        if (i == 0 || str[i] != str[i - 1]) {
            count = 1; // 如果当前字符和前一个字符不同，重置计数器
        } else {
            count++; // 如果当前字符和前一个字符相同，增加计数器
        }

        int index = str[i] - 'A'; // 将字符转换为索引（0-25）
        if (count > charMaxCount[index]) {
            charMaxCount[index] = count; // 更新字符的最大连续出现次数
        }
    }

    // 将连续出现次数存储在数组中，并排序
    int counts[26], j = 0;
    for (int i = 0; i < 26; i++) {
        if (charMaxCount[i] > 0) {
            counts[j++] = charMaxCount[i];
        }
    }

    // 冒泡排序
```

```
    for (int i = 0; i < j - 1; i++) {
        for (int p = 0; p < j - i - 1; p++) {
            if (counts[p] < counts[p + 1]) {
                int temp = counts[p];
                counts[p] = counts[p + 1];
                counts[p + 1] = temp;
            }
        }
    }

    // 输出结果
    if (k > j) {
        printf("-1\n"); // 如果k大于数组长度，输出-1
    } else {
        printf("%d\n", counts[k - 1]); // 输出第k多的字符的次数
    }

    return 0;
}
```

完整用例

用例1

```
AAAAHHHBBCDHHHH
3
```

用例2

```
ABCDDEFG
1
```

用例3

```
ABC
4
```

用例4

```
HHHHHHHHH
5
```

用例5

```
AAABBBCCC
2
```

用例6

```
ABBBCCCCCDDDDEEEEEE
4
```

用例7

```
AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQRRSSTTUUVVWWXXYYZZ
10
```

用例8

```
AAABBBCCCDDDEEEFFFGGGHHHIIIIJJJKKKLLLMMMNNNNOOO
3
```

用例9

```
AAAABBBBCCCCDDDDDEEEEEFFFFFGGGGGHHHHHIIIIIJJJJJ
2
```

用例10

```
ABCDEFGHHHHHHIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
1
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师