【华为OD机考 统一考试机试C卷】字符串分割转换（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

给定一个非空字符串S，其被N个'-'分隔成N+1的子串，给定正整数K，要求除第一个子串外，其余的子串每K个字符组成新的子串，并用'-'分隔。
对于新组成的每一个子串，如果它含有的小写字母比大写字母多，则将这个子串的所有大写字母转换为小写字母；
反之，如果它含有的大写字母比小写字母多，则将这个子串的所有小写字母转换为大写字母；大小写字母的数量相等时，不做转换。

输入描述

输入为两行，第一行为参数K，第二行为字符串S。

输出描述

输出转换后的字符串。

用例1

输入

```
3
12abc-abCABc-4aB@
```

输出

```
12abc-abc-ABC-4aB-@
```

说明

> 子串为12abc、abCABc、4aB@，第一个子串保留，
> 后面的子串每3个字符一组为abC、ABc、4aB、@，
> abC中小写字母较多，转换为abc，
> ABc中大写字母较多，转换为ABC，
> 4aB中大小写字母都为1个，不做转换，
> @中没有字母，连起来即12abc-abc-ABC-4aB-@

用例2

输入

```
12
12abc-abCABc-4aB@
```

输出

```
12abc-abCABc4aB@
```

说明

> 子串为12abc、abCABc、4aB@，第一个子串保留，
> 后面的子串每12个字符一组为abCABc4aB@，
> 这个子串中大小写字母都为4个，不做转换，
> 连起来即12abc-abCABc4aB@

C++

```cpp
#include <iostream>
#include <sstream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int k;
    string s;
    getline(cin, s);
    k = stoi(s); // 输入正整数K
    getline(cin, s); // 输入字符串S
    vector<string> sArr;
    size_t pos = 0;
    while ((pos = s.find("-")) != string::npos) { // 将S按照“-”分隔成N+1个子串
        string token = s.substr(0, pos);
        sArr.push_back(token);
        s.erase(0, pos + 1);
    }
    sArr.push_back(s);
    string prefix = sArr[0]; // 第一个子串
    string postfix; // 除第一个子串外的所有子串
    for (int i = 1; i < sArr.size(); i++) {
        postfix += sArr[i];
    }

    vector<char> postfixChars(postfix.begin(), postfix.end());

    string newS = "";
    for (int i = 0; i < postfixChars.size(); i++) {
        if ((i + 1) % k == 0 && i + 1 != postfixChars.size()) {
            newS += postfixChars[i];
            newS += '-';
        } else {
            newS += postfixChars[i];
        }
    }

  vector<string> newSArr;
    pos = 0;
```

```
    while ((pos = newS.find("-")) != string::npos) { // 将新组成的每一个子串按照“-”分隔
        string token = newS.substr(0, pos);
        newSArr.push_back(token);
        newS.erase(0, pos + 1);
    }
    newSArr.push_back(newS);

    string result = "";
    for (string str : newSArr) {
        int upperCase = count_if(str.begin(), str.end(), [](char c) { return isupper(c); });
        int lowerCase = count_if(str.begin(), str.end(), [](char c) { return islower(c); });

        if (upperCase > lowerCase) {
            transform(str.begin(), str.end(), str.begin(), ::toupper);
            result += str;
            result += '-';
        } else if (lowerCase > upperCase) {
            transform(str.begin(), str.end(), str.begin(), ::tolower);
            result += str;
            result += '-';
        } else {
            result += str;
            result += '-';
        }
    }

    result = result.substr(0, result.length() - 1);

    cout << prefix << "-" << result << endl;

    return 0;
}
```

java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int k = Integer.parseInt(scanner.nextLine()); // 输入正整数K
        String s = scanner.nextLine(); // 输入字符串S
        String[] sArr = s.split("-"); // 将S按照“-”分隔成N+1个子串
        String prefix = sArr[0]; // 第一个子串
        StringBuilder postfixSb = new StringBuilder(); // 除第一个子串外的所有子串
        for (int i = 1; i < sArr.length; i++) {
            postfixSb.append(sArr[i]);
        }
        char[] postfixChars = postfixSb.toString().toCharArray(); // 将除第一个子串外的所有子串拼接成字符数组
        StringBuilder newSb = new StringBuilder();
        for (int i = 0; i < postfixChars.length; i++) { // 将除第一个子串外的所有子串每K个字符组成新的子串，并用‘-’分隔
            if ((i + 1) % k == 0 && i + 1 != postfixChars.length) {
                newSb.append(postfixChars[i]).append("-");
            } else {
                newSb.append(postfixChars[i]);
            }
        }
        String[] newSArr = newSb.toString().split("-"); // 将新组成的每一个子串按照“-”分隔
        StringBuilder resultSb = new StringBuilder();

```

```
        for (String str : newSArr) { // 对于新组成的每一个子串，如果它含有的小写字母比大写字母多，则将这个子串的所有大写字母转换为小写字母；反之，如果它含有的大写字母比小写字母多，则将这个子串的所有小写字母转换为大写字母；大小写字母的数量相等时，不做转换

            long upperCase = str.chars().filter(Character::isUpperCase).count();

            long lowerCase = str.chars().filter(Character::isLowerCase).count();
            if (upperCase > lowerCase) {
                resultSb.append(str.toUpperCase()).append("-");
            } else if (lowerCase > upperCase) {
                resultSb.append(str.toLowerCase()).append("-");
            } else {
                resultSb.append(str).append("-");
            }
        }
        String postfix = resultSb.toString().substring(0, resultSb.length() - 1); // 将处理后的新组成的每一个子串按照“-”拼接成字符串
        System.out.println(prefix + "-" + postfix); // 输出转换后的字符串
    }
}
```

python

```
k = int(input()) # 输入正整数K
s = input() # 输入字符串S
sArr = s.split("-") # 将S按照“-”分隔成N+1个子串
prefix = sArr[0] # 第一个子串
postfixSb = "" # 除第一个子串外的所有子串
for i in range(1, len(sArr)):
    postfixSb += sArr[i]
postfixChars = list(postfixSb) # 将除第一个子串外的所有子串拼接成字符数组
newSb = ""
for i in range(len(postfixChars)): # 将除第一个子串外的所有子串每K个字符组成新的子串，并用‘-’分隔
    if (i + 1) % k == 0 and i + 1 != len(postfixChars):
        newSb += postfixChars[i] + "-"
    else:
        newSb += postfixChars[i]
newSArr = newSb.split("-") # 将新组成的每一个子串按照“-”分隔
resultSb = ""
for str in newSArr: # 对于新组成的每一个子串，如果它含有的小写字母比大写字母多，则将这个子串的所有大写字母转换为小写字母；反之，如果它含有的大写字母比小写字母多，则将这个子串的所有小写字母转换为大写字母；大小写字母的数量相等时，不做转换
    upperCase = sum(1 for c in str if c.isupper())
    lowerCase = sum(1 for c in str if c.islower())
    if upperCase > lowerCase:
        resultSb += str.upper() + "-"
    elif lowerCase > upperCase:
        resultSb += str.lower() + "-"
    else:
        resultSb += str + "-"
postfix = resultSb[:-1] # 将处理后的新组成的每一个子串按照“-”拼接成字符串
print(prefix + "-" + postfix) # 输出转换后的字符串


```

JavaScript

```
const readline = require("readline");

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

```

```
const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const k = parseInt(lines[0]);
    const s = lines[1];
     const arr = s.split("-");
  const prefix = arr.shift();
  const postfix = arr
    .join("")
    .match(new RegExp(`.{1,${k}}`, "g"))
    .map((str) => {
      let upperCase = 0;
      let lowerCase = 0;
      [...str].forEach((char) => {
        if (/[a-z]/.test(char)) lowerCase++;
        if (/[A-Z]/.test(char)) upperCase++;
      });
      if (upperCase > lowerCase) {
        return str.toUpperCase();
      }
      if (lowerCase > upperCase) {
        return str.toLowerCase();
      }
      return str;
    })
    .join("-");


    console.log(prefix + "-" + postfix);
  }
});

```

C语言

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

#define MAX_LEN 500001 // 假设字符串的最大长度不超过500000

// 函数：将子串中的字母根据大小写字母数量转换为全大写或全小写
void convert(char *str) {
    int upper = 0, lower = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (isupper(str[i])) upper++;
        if (islower(str[i])) lower++;
    }
    for (int i = 0; str[i] != '\0'; i++) {
        if (upper > lower) str[i] = toupper(str[i]);
        if (lower > upper) str[i] = tolower(str[i]);
    }
}

int main() {
    int k; // 输入正整数K

```

```
    scanf("%d\n", &k);

    char s[MAX_LEN]; // 输入字符串S
    fgets(s, MAX_LEN, stdin);
    s[strcspn(s, "\n")] = 0; // 去除换行符

    char *token = strtok(s, "-"); // 第一个子串
    printf("%s", token); // 输出第一个子串

    char postfix[MAX_LEN] = ""; // 除第一个子串外的所有子串
    while ((token = strtok(NULL, "-")) != NULL) {
        strcat(postfix, token);
    }

    char newStr[MAX_LEN] = ""; // 新组成的字符串
    int len = strlen(postfix);
    for (int i = 0; i < len; i++) {
        strncat(newStr, &postfix[i], 1); // 逐个字符拼接到新字符串中
        if ((i + 1) % k == 0 && i + 1 != len) {
            strcat(newStr, "-"); // 每K个字符后添加分隔符
        }
    }

    char *newToken = strtok(newStr, "-"); // 按照“-”分隔新组成的字符串
    while (newToken != NULL) {
        convert(newToken); // 转换子串中的字母
        printf("-%s", newToken); // 输出转换后的子串
        newToken = strtok(NULL, "-"); // 继续获取下一个子串
    }

    return 0;
}
```

完整用例

用例1

```
3
12abc-abCABc-4aB@
```

用例2

```
2
aBcDe-FgHiJ-KlMnO
```

用例3

```
4
aBcDe-FgHiJ-KlMnO
```

用例4

```
5
aBcDe-FgHiJ-KlMnO
```

用例5

```
1
aBcDe-FgHiJ-KlMnO
```

用例6

```
6
aBcDe-FgHiJ-KlMnO
```

用例7

```
3
aBcDe-FgHiJ-KlMnO
```

用例8

```
2
aBcDe-FgHiJ-KlMnO
```

用例9

```
4
aBcDe-FgHiJ-KlMnO
```

用例10

```
5
aBcDe-FgHiJ-KlMnO
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师