【模拟】-字符串分割（二）

题目描述与示例

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。

示例一

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-@`

示例二

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@` 。

示例三

输入

```
5
12abc-abCABc-4aB@
```

输出

```
12abc-ABCAB-c4ab@
```

解题思路

题意理解

纯字符串模拟题，依据题目含义进行相应模拟即可。要注意以下几点处理：

1. 第一个字符串不用做任何修改
2. 最后一个字符串如果长度不足 `K` ，则保留原长度，在计算中要防止越界操作（尤其是使用非Python语言的同学）

将这整个问题分为三个小问题来解决即可：

1. 如何让初始字符串的分割和合并
2. 如何每 `K` 个字符一组构成子字符串
3. 如何对每个子字符串进行判断和处理

有很多同学对题意提出了不同的理解。对于

> 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。

这个条件，有部分同学认为其意思是对切割后的单个子字符串，而不是所有子字符串（除第一个外）构成的新字符串，进行 `K` 个 `K` 个的分组。

即对于例子

```
5
12abc-abCABc-4aB@
```

部分同学认为它的分组应该为

```
1 12abc abCAB c 4aB@
```

这确实是出题人糟糕的题目描述会带来的题意理解混淆。

遇到这种情况的时候，我们必须根据示例出发来反推题意。

如果是按照这种错误理解方式来进行的话，那么答案应该为

```
1 12abc-ABCAB-c-4aB@
```

而不是示例输出的

```
1 12abc-ABCAB-c4ab@
```

对于示例二进行同样的推理，我们也能够得知如果是错误的理解方式，其答案会是

```
1 12abc-abCABc-4aB@
```

而不是正确答案

```
1 12abc-abCABc4aB@
```

> 这个问题实际上包含了非常重要的启示
>
> - 既然这是考试，我们就无法决定出题人的行为，即使他们的出题水平非常差，我们也没有办法，只能够去顺应这样的结果。
> - 在糟糕的文字表述之下，我们能做的，除了进一步提高自己对文字的阅读理解能力之外，还需要提高根据用例以及其备注，进行题意理解反推的能力，这样才能够以不变应万变。

初始字符串的分割和合并

初始字符串的分割非常简单，我们知道分割符为 `"-"` ，故我们可以直接对输入的字符串进行分割，将其分割为包含若干个子字符串的列表 `lst` ，其代码为

```
lst = input().split("-")
```

接下来，我们需要把除了第一个子字符串之外的其他剩余所有字符串合并为 `s` ，方便后续操作。注意这里的合并符直接选择空字符串 `""` 即可，其代码为

```
s = "".join(lst[1:])
```

每 `K` 个字符一组构成子字符串

接下来，我们希望能够对 `s` 里的字符以 `K` 为长度进行分组，得到每一个子字符串 `sub_s` 。

我们可以在 `range()` 中选择步长为 `K` 来遍历每一个子字符串的起始索引 `idx` 。

显然， `idx` 的取值就会是 `0, K, 2*K, 3*K...` 以此类推。

在Python中，我们可以通过切片来获得一个字符串的长度为 `K` 的子字符串，即 `sub_s = s[idx:idx+K]` 。

注意到，如果 `len(s)` 如果不是恰好是 `K` 的倍数的话，最后一个子字符串的长度是不足 `K` 的。

根据题意，我们需要保留最后一个子字符串的原长度。

但Python的切片是非常宽容的。当我们选择 `sub_s = s[idx:idx+K]` 时，如果 `idx+K` 大于字符串 `s` 的长度 `len(s)` ，也不会发生报错。这个切片会直接取完原字符串 `s` 的剩余部分，用 `len(s)` 来作为终止位置。

故该过程的代码为

```
for idx in range(0, len(s), K):
    sub_s = s[idx:idx+K]
    ans.append(get_res(sub_s))
```

如果你对 `sub_s = s[idx:idx+K]` 的越界问题实在不放心，或者你是使用Java或者C++等其他语言，可以将终止位置 `idx+K` 替换为 `min(idx+K, len(s))` ，这样就能够确保切片的终止位置，不会比字符串长度 `len(s)` 更大。

其代码为

```
for idx in range(0, len(s), K):
    sub_s = s[idx:min(idx+K, len(s))]
    ans.append(get_res(sub_s))
```

子字符串的判断和处理

对于上述得到的每一个子字符串 `sub_s` ，我们都需要去进行判断和修改。

这是一个额外的子问题，可以抽象成 `get_res()` 来处理。

我们可以用两个变量 `upper_num` 和 `lower_num` 来分别记录 `sub_s` 中大写、小写的字母的个数。

```
upper_num = 0
lower_num = 0
```

接着遍历 `sub_s` 中的每一个字符 `ch` ，如果

- `ch` 是小写字母，则 `lower_num` 的个数增加
- `ch` 是大写字母，则 `upper_num` 的个数增加
- `ch` 是其他无关字符，则不做任何修改

注意，字符 `ch` 的大小写判断可以直接使用字符串方法 `islower()` 和 `isupper()` 来实现，也可以通过字符串比较的方式 `"a" <= ch <= "z"` 和 `"A" <= ch <= "Z"` 来实现。

```
for ch in sub_s:
    if ch.islower():
        lower_num += 1
    if ch.isupper():
        upper_num += 1
```

退出循环后，我们判断两个变量 `upper_num` 和 `lower_num` 之间的大小关系。若

- 大小写字母个数相等，则返回原子字符串 `sub_s`
- 大写字母更多，返回将 `sub_s` 中所有小写字母都改为大写的字符串，使用字符串方法 `upper()` 实现
- 小写字母更多，返回将 `sub_s` 中所有大写字母都改为小写的字符串，使用字符串方法 `lower()` 实现

```
if lower_num == upper_num:
    return sub_s
return sub_s.lower() if lower_num > upper_num else sub_s.upper()
```

故 `get_res()` 的整体代码为

```
def get_res(sub_s):
    upper_num = 0
    lower_num = 0

    for ch in sub_s:
        if ch.islower():
            lower_num += 1
        if ch.isupper():
            upper_num += 1

    if lower_num == upper_num:
        return sub_s
    return sub_s.lower() if lower_num > upper_num else sub_s.upper()
```

那么在主函数代码中，我们通过将得到的每一个子字符串 `sub_s` 传入函数 `get_res()` ，就可以获得每一个子字符串处理后的结果了。即 `ans.append(get_res(sub_s))` 。

代码

Python

```python
# 题目：2024E-字符串分割（二）
# 分值：100
# 作者：许老师-闭着眼睛学数理化
# 算法：模拟/字符串
# 代码看不懂的地方，请直接在群上提问


# 根据sub_s中的小写字母、大写字母个数，对sub_s进行修改的函数
def get_res(sub_s):
    # 两个变量分别记录sub_s中大写、小写字母的个数
    upper_num = 0
    lower_num = 0
    for ch in sub_s:
        if ch.islower():
            lower_num += 1
        if ch.isupper():
            upper_num += 1
    # 若大小写字母个数相等，则返回原子字符串sub_s
    if lower_num == upper_num:
        return sub_s
    # 否则，根据大小关系，返回sub_s.lower()或sub_s.upper()
    return sub_s.lower() if lower_num > upper_num else sub_s.upper()


K = int(input())
# 对原始字符串根据分割符"-"进行切割
lst = input().split("-")

# 对除了第一个字符串以外的其他字符串进行合并
# 构成一个新的字符串s
s = "".join(lst[1:])

ans = list()
```

```python

# 对于字符串s，每K个字符进行切片操作
for idx in range(0, len(s), K):
    # 获得长度为K的子字符串，注意最后一个子字符串的终止位置不能超过len(s)
    sub_s = s[idx:min(idx+K, len(s))]
    # 调用get_res(sub_s)，将结果存入ans中
    ans.append(get_res(sub_s))

# 输出答案，注意此处需要加上第一个字符串
print("-".join([lst[0]] + ans))
```

Java

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {
    // 根据sub_s中的小写字母、大写字母个数，对sub_s进行修改的函数
    public static String getRes(String sub_s) {
        // 两个变量分别记录sub_s中大写、小写字母的个数
        int upper_num = 0;
        int lower_num = 0;
        for (char ch : sub_s.toCharArray()) {
            if (Character.isLowerCase(ch)) {
                lower_num++;
            }
            if (Character.isUpperCase(ch)) {
                upper_num++;
            }
        }
        // 若大小写字母个数相等，则返回原子字符串sub_s
        if (lower_num == upper_num) {
            return sub_s;
        }
        // 否则，根据大小关系，返回sub_s.toLowerCase()或sub_s.toUpperCase()
        return lower_num > upper_num ? sub_s.toLowerCase() :
sub_s.toUpperCase();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int K = scanner.nextInt();
        scanner.nextLine();  // Consume newline left-over
        // 对原始字符串根据分割符"-"进行切割
```

```java
        String[] lst = scanner.nextLine().split("-");

        // 对除了第一个字符串以外的其他字符串进行合并
        // 构成一个新的字符串s
        StringBuilder s = new StringBuilder();
        for (int i = 1; i < lst.length; i++) {
            s.append(lst[i]);
        }

        List<String> ans = new ArrayList<>();

        // 对于字符串s，每K个字符进行切片操作
        for (int idx = 0; idx < s.length(); idx += K) {
            // 获得长度为K的子字符串，注意最后一个子字符串的终止位置不能超过len(s)
            String sub_s = s.substring(idx, Math.min(idx + K, s.length()));
            // 调用getRes(sub_s)，将结果存入ans中
            ans.add(getRes(sub_s));
        }

        // 输出答案，注意此处需要加上第一个字符串
        ans.add(0, lst[0]);
        System.out.println(String.join("-", ans));
    }
}

```

C++

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

using namespace std;

// 根据sub_s中的小写字母、大写字母个数，对sub_s进行修改的函数
string getRes(const string& sub_s) {
    // 两个变量分别记录sub_s中大写、小写字母的个数
    int upper_num = 0;
    int lower_num = 0;
    for (char ch : sub_s) {
        if (islower(ch)) {
            lower_num++;
        }
        if (isupper(ch)) {
```

```
            upper_num++;
        }
    }
    // 若大小写字母个数相等，则返回原子字符串sub_s
    if (lower_num == upper_num) {
        return sub_s;
    }
    // 否则，根据大小关系，返回sub_s变小写或大写
    string result = sub_s;
    if (lower_num > upper_num) {
        transform(result.begin(), result.end(), result.begin(), ::tolower);
    } else {
        transform(result.begin(), result.end(), result.begin(), ::toupper);
    }
    return result;
}

int main() {
    int K;
    cin >> K;
    cin.ignore();  // 忽略换行符

    // 对原始字符串根据分割符"-"进行切割
    string input;
    getline(cin, input);

    vector<string> lst;
    size_t pos = 0;
    string token;
    while ((pos = input.find('-')) != string::npos) {
        token = input.substr(0, pos);
        lst.push_back(token);
        input.erase(0, pos + 1);
    }
    lst.push_back(input);

    // 对除了第一个字符串以外的其他字符串进行合并
    // 构成一个新的字符串s
    string s;
    for (size_t i = 1; i < lst.size(); i++) {
        s += lst[i];
    }

    vector<string> ans;

    // 对于字符串s，每K个字符进行切片操作
    for (size_t idx = 0; idx < s.size(); idx += K) {
```

```
65          // 获得长度为K的子字符串，注意最后一个子字符串的终止位置不能超过len(s)
66          string sub_s = s.substr(idx, min(K, (int)(s.size() - idx)));
67          // 调用getRes(sub_s)，将结果存入ans中
68          ans.push_back(getRes(sub_s));
69      }
70
71      // 输出答案，注意此处需要加上第一个字符串
72      ans.insert(ans.begin(), lst[0]);
73      for (size_t i = 0; i < ans.size(); i++) {
74          if (i != 0) cout << "-";
75          cout << ans[i];
76      }
77      cout << endl;
78
79      return 0;
80  }
81
```

时空复杂度

时间复杂度：`O(N)`。

空间复杂度：`O(1)`。