题目描述

给定一个由多个命令字组成的命令字符串：
1、字符串长度小于等于127字节，只包含大小写字母，数字，下划线和偶数个双引号；
2、命令字之间以一个或多个下划线_进行分割；
3、可以通过两个双引号""来标识包含下划线_的命令字或空命令字（仅包含两个双引号的命令字），双引号不会在命令字内部出现；

请对指定索引的敏感字段进行加密，替换为******（6个*），并删除命令字前后多余的下划线_。
如果无法找到指定索引的命令字，输出字符串ERROR。

输入描述

输入为两行，第一行为命令字索引K（从0开始），第二行为命令字符串S。

输出描述

输出处理后的命令字符串，如果无法找到指定索引的命令字，输出字符串ERROR

用例

输入	1 password__a12345678_timeout_100
输出	password_******_timeout_100

题目描述

给定一个由多个命令字组成的命令字符串：
1、字符串长度小于等于127字节，只包含大小写字母，数字，下划线和偶数个双引号；
2、命令字之间以一个或多个下划线_进行分割；
3、可以通过两个双引号""来标识包含下划线_的命令字或空命令字（仅包含两个双引号的命令字），双引号不会在命令字内部出现；

请对指定索引的敏感字段进行加密，替换为******（6个*），并删除命令字前后多余的下划线_。
如果无法找到指定索引的命令字， 输出字符串ERROR。

输入描述

输入为两行，第一行为命令字索引K（从0开始），第二行为命令字符串S。

输出描述

输出处理后的命令字符串，如果无法找到指定索引的命令字，输出字符串ERROR

用例

输入	1 password__a12345678_timeout_100
输出	password_******_timeout_100
说明	无

输入	2 aaa_password_"a12_45678"_timeout__100_""_
输出	aaa_password_******_timeout_100_""
说明	无

题目解析

一开始我想构造一个正则只匹配“_”，而不会收到引号的影响，但是发现很难。

题目解析

一开始我想构造一个正则只匹配“_”，而不会收到引号的影响，但是发现很难。

于是，就想到之前有一题报文解压缩也是处理字符串，但是正则不好构造，后面使用栈结构来解决的。

那么这题是否也可以用栈结构来解题呢?

答案是可以的。逻辑如下：

创建一个stack栈，用于接收输入字符串遍历出来的每一个字符

- 如果是'_'字符，则看看stack栈底是否为'"'，如果是，则说明当前‘_’字符是命令字的组成部分，而不是命令字分隔符，因此保留push进栈。如果不是，则说明当前'_'字符是分隔符，我们将stack栈中所有元素拼接为字符串保存进result，然后stack.length = 0，将栈清空。
- 如果是'"'，则看看当前栈是否为空，若为空，则说明当前‘”’是第一个引号，若不为空，则说明当前引号是第二个引号，此时我们需要将stack栈中所有元素拼接为字符串保存进result，然后stack.length = 0，将栈清空。
- 如果既不是‘_’字符，也不是'"'字符，则是普通字符，直接push进stack

遍历完所有字符后，则判断stack是否为空，若不为空，则还需要将stack栈中所有元素拼接为字符串保存进result。

当输入字符串,如
aaa_password_"a12_45678"_timeout__100_""_

经过上面逻辑处理后，result内容如下

[

'aaa',

'password',

'"a12_45678"',

'timeout',

'',

'100',

'""',

''

]

可以发现，result中存在不少空串，因此我们需要过滤掉空串

[

'aaa',

'password',

'"a12_45678"',

'timeout',

'100',

'""'

]

此时找到索引K对应的元素，将其替换为******

[

'aaa',

'password',

'******',

'timeout',

'100',

''''

]

最后以‘_’连接result中的元素为字符串返回，就是题解

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
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];

    console.log(encryptSensitive(s, k));

    lines.length = 0;
  }
});

function encryptSensitive(s, k) {
  let stack = [];
  let result = [];

  for (let i = 0; i < s.length; i++) {
    if (s[i] === "_" && stack[0] !== '"') {
      result.push(stack.join(""));
      stack.length = 0;
    } else if (s[i] === '"' && stack.length !== 0) {
      result.push(stack.join("") + '"');
      stack.length = 0;
```

```
      stack.length = 0;
    } else {
      stack.push(s[i]);
    }
  }
  if (stack.length) result.push(stack.join(""));

  result = result.filter((ele) => ele !== "");

  if (k > result.length - 1) return "ERROR";

  result[k] = "******";

  return result.join("_");
}
```

Java算法源码

```
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;
import java.util.StringJoiner;
import java.util.stream.Collectors;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int k = Integer.parseInt(sc.nextLine());
    String s = sc.nextLine();

    System.out.println(getResult(k, s));
  }

  public static String getResult(int k, String s) {
    StringBuilder stack = new StringBuilder();
    LinkedList<String> result = new LinkedList<>();

    for (int i = 0; i < s.length(); i++) {
      char c = s.charAt(i);

      if (c == '_' && (stack.length() == 0 || stack.char
        result.add(stack.toString());
```

```
            result.add(stack.toString());
            stack = new StringBuilder();
        } else if (c == '"' && stack.length() != 0) {
            stack.append('"');
            result.add(stack.toString());
            stack = new StringBuilder();
        } else {
            stack.append(c);
        }
    }

    if (stack.length() > 0) result.add(stack.toString())

    List<String> ans = result.stream().filter(str -> !"

    if (k > ans.size() - 1) return "ERROR";
    ans.set(k, "******");

    StringJoiner sj = new StringJoiner("_");
    for (String an : ans) sj.add(an);
    return sj.toString();
  }
}
```

Python算法源码

复制

```
# 输入获取
k = int(input())
s = input()


# 算法入口
def getResult(k, s):
    stack = []
    res = []

    for c in s:
        if c == '_' and (len(stack) == 0 or stack[0] !=
            res.append("".join(stack))
            stack = []
        elif c == '"' and len(stack) != 0:
            stack.append(c)
            res.append("".join(stack))
```

```
            res.append("".join(stack))
            stack = []
        elif c == '"' and len(stack) != 0:
            stack.append(c)
            res.append("".join(stack))
            stack = []
        else:
            stack.append(c)

    if len(stack) > 0:
        res.append("".join(stack))

    ans = list(filter(lambda x: x != "", res))

    if k > len(ans) - 1:
        return "ERROR"

    ans[k] = "******"

    return "_".join(ans)


# 算法调用
print(getResult(k, s))
```