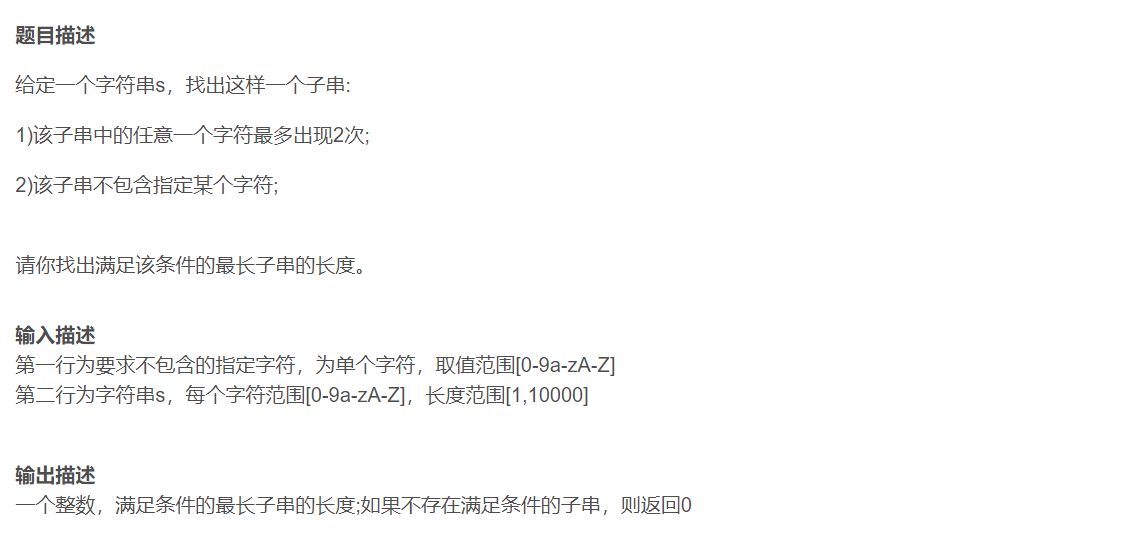
# **E卷-寻找符合要求的最长子串[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**







import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

char ch = scanner.nextLine().charAt(0);

String s = scanner.nextLine();

int[] cnt = new int[200]; // 初始化计数数组，用于记录每个字符在当前子串中的出现次数

int lc = 0; // 左指针，表示当前子串的起始位置

int res = 0; // 用于存储满足条件的最长子串的长度

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

if (s.charAt(i) == ch) { // 如果当前字符是指定的字符

lc = i + 1; // 更新左指针，跳过当前字符

cnt = new int[200]; // 重置计数数组

} else {

cnt[s.charAt(i)]++; // 增加当前字符的计数

while (cnt[s.charAt(i)] > 2) { // 如果当前字符在子串中出现超过2次

cnt[s.charAt(lc)]--; // 移动左指针并减少对应字符的计数

lc++; // 移动左指针，缩小子串范围

}

res = Math.max(res, i - lc + 1); // 更新最长子串的长度

}

}

System.out.println(res); // 输出满足条件的最长子串的长度

scanner.close();

}

}



ch = input() # 读取指定的字符

s = input() # 读取输入字符串

cnt = [0] \* 200 # 初始化计数数组，用于记录每个字符在当前子串中的出现次数

lc = 0 # 左指针，表示当前子串的起始位置

res = 0 # 用于存储满足条件的最长子串的长度

for i in range(len(s)):

if s[i] == ch: # 如果当前字符是指定的字符

lc = i + 1 # 更新左指针，跳过当前字符

cnt = [0] \* 200 # 重置计数数组

else:

cnt[ord(s[i])] += 1 # 增加当前字符的计数

while cnt[ord(s[i])] > 2: # 如果当前字符在子串中出现超过2次

cnt[ord(s[lc])] -= 1 # 移动左指针并减少对应字符的计数

lc += 1 # 移动左指针，缩小子串范围

res = max(res, i - lc + 1) # 更新最长子串的长度

print(res) # 输出满足条件的最长子串的长度



#include <iostream>

#include <vector>

using namespace std;

int main() {

char ch;

string s;

cin >> ch;

cin >> s;

vector<int> cnt(200, 0); // 初始化计数数组，用于记录每个字符在当前子串中的出现次数

int lc = 0; // 左指针，表示当前子串的起始位置

int res = 0; // 用于存储满足条件的最长子串的长度

for (int i = 0; i < s.size(); i++) {

if (s[i] == ch) { // 如果当前字符是指定的字符

lc = i + 1; // 更新左指针，跳过当前字符

cnt = vector<int>(200, 0); // 重置计数数组

} else {

cnt[s[i]]++; // 增加当前字符的计数

while (cnt[s[i]] > 2) { // 如果当前字符在子串中出现超过2次

cnt[s[lc]]--; // 移动左指针并减少对应字符的计数

lc++; // 移动左指针，缩小子串范围

}

res = max(res, i - lc + 1); // 更新最长子串的长度

}

}

cout << res << endl; // 输出满足条件的最长子串的长度

return 0;

}



#include <stdio.h>

#include <string.h>

int main() {

char ch;

char s[1000005]; //数组大小开大，如果段错误就继续开大一点

int cnt[200] = {0}; // 初始化计数数组，用于记录每个字符在当前子串中的出现次数

int lc = 0; // 左指针，表示当前子串的起始位置

int res = 0; // 用于存储满足条件的最长子串的长度

scanf("%c", &ch);

scanf("%s", s);

int len = strlen(s);

for (int i = 0; i < len; i++) {

if (s[i] == ch) { // 如果当前字符是指定的字符

lc = i + 1; // 更新左指针，跳过当前字符

for (int j = 0; j < 200; j++) cnt[j] = 0; // 重置计数数组

} else {

cnt[(int)s[i]]++; // 增加当前字符的计数

while (cnt[(int)s[i]] > 2) { // 如果当前字符在子串中出现超过2次

cnt[(int)s[lc]]--; // 移动左指针并减少对应字符的计数

lc++; // 移动左指针，缩小子串范围

}

if (res < i - lc + 1) res = i - lc + 1; // 更新最长子串的长度

}

}

printf("%d\n", res); // 输出满足条件的最长子串的长度

return 0;

}



const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

let input = [];

rl.on('line', (line) => {

input.push(line);

}).on('close', () => {

const ch = input[0];

const s = input[1];

let cnt = Array(200).fill(0); // 初始化计数数组，用于记录每个字符在当前子串中的出现次数

let lc = 0; // 左指针，表示当前子串的起始位置

let res = 0; // 用于存储满足条件的最长子串的长度

for (let i = 0; i < s.length; i++) {

if (s[i] === ch) { // 如果当前字符是指定的字符

lc = i + 1; // 更新左指针，跳过当前字符

cnt = Array(200).fill(0); // 重置计数数组

} else {

cnt[s.charCodeAt(i)]++; // 增加当前字符的计数

while (cnt[s.charCodeAt(i)] > 2) { // 如果当前字符在子串中出现超过2次

cnt[s.charCodeAt(lc)]--; // 移动左指针并减少对应字符的计数

lc++; // 移动左指针，缩小子串范围

}

res = Math.max(res, i - lc + 1); // 更新最长子串的长度

}

}

console.log(res); // 输出满足条件的最长子串的长度

});



package main

import (

"bufio"

"fmt"

"os"

)

func main() {

reader := bufio.NewReader(os.Stdin)

ch, \_ := reader.ReadByte()

reader.ReadByte() // 吞掉换行符

s, \_ := reader.ReadString('\n')

s = s[:len(s)-1] // 去掉末尾的换行符

cnt := make([]int, 200) // 初始化计数数组，用于记录每个字符在当前子串中的出现次数

lc := 0 // 左指针，表示当前子串的起始位置

res := 0 // 用于存储满足条件的最长子串的长度

for i := 0; i < len(s); i++ {

if s[i] == ch { // 如果当前字符是指定的字符

lc = i + 1 // 更新左指针，跳过当前字符

cnt = make([]int, 200) // 重置计数数组

} else {

cnt[s[i]]++ // 增加当前字符的计数

for cnt[s[i]] > 2 { // 如果当前字符在子串中出现超过2次

cnt[s[lc]]-- // 移动左指针并减少对应字符的计数

lc++ // 移动左指针，缩小子串范围

}

if res < i-lc+1 {

res = i - lc + 1 // 更新最长子串的长度

}

}

}

fmt.Println(res) // 输出满足条件的最长子串的长度

}