【不定滑窗】-寻找符合要求的最长子串

题目描述与示例

题目描述

给定一个字符串 `s` ，找出这样一个子串：

1. 该子串中的任意一个字符最多出现 `2` 次；
2. 该子串不包含指定某个字符；

请你找出满足该条件的最长子串的长度。

输入

第一行为要求不包含的指定字符，为单个字符，取值范围 `[0-9a-zA-Z]`

第二行为字符串 s，每个字符范围 `[0-9a-zA-Z]` ，长度范围 `[1,10000]`

输出

一个整数，满足条件的最长子串的长度；

如果不存在满足条件的子串，则返回 `0`

示例一

输入

```
D
ABC123
```

输出

```
6
```

示例二

输入

```
D
ABACD1231
```

输出

```
4
```

解题思路

子串被两个条件限制：

1. 同一字符数目不能超过 `2`
2. 不能包含指定字符

如果只考虑第一个限制条件，那么这就是一道典型的滑动窗口题目。该子问题用我们的**滑窗三问三答**来解题。

滑窗三问

Q1：对于每一个右指针 `right` 所指的元素 `ch`，做什么操作？

Q2：什么时候要令左指针 `left` 右移？ `left` 对应的元素做什么操作？ `while` 中的**循环不变量**是什么？

Q3：什么时候进行 `ans` 的更新？

滑窗三答

A1：将 `ch` 加入哈希表 `hash_table` 中进行频数统计中

A2：右指针 `right` 所指的元素 `ch` 在哈希表中出现的个数超过了 `2`，说明当前滑窗对应的子串不符合题意，需要将 `left` 所指的元素 `s[left]` 在哈希表中的频数 `-1`，同时令 `left` 右移，直到 `ch` 在哈希集合中的频数为 `2`。

A3： `left` 右移结束后，更新 `ans`。

对于第二个限制条件，其实也很好处理。有两种处理方式：

1. 在遍历 `s` 的过程中，遇到指定字符 `a` 直接跳过，从 `right+1` 位置开始重新进行滑窗过程，同时哈希表清空
2. 先根据指定字符 `a` 对 `s` 进行切割，得到单词列表 `lst`，对于列表中的每一个单词 `word` 分别进行滑窗过程，对所有滑窗过程得到结果 `temp` 取最大值。

代码

解法一

（对第二个限制条件的第一种处理方式）

Python

```python
# 题目：2024E-寻找符合要求的最长子串
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：不定滑窗
# 代码看不懂的地方，请直接在群上提问

import collections

a = input()
s = input()

left, ans = 0, 0
hash_table = collections.Counter()      # 哈希表记录窗口中各个字符出现频数
for right, ch in enumerate(s):
    if ch != a:                         # ch不为应该排除的数：进行滑窗
        hash_table[ch] += 1             # A1
        while (hash_table[ch] == 3):    # A2
            hash_table[s[left]] -= 1
            left += 1
        ans = max(ans, right-left+1)    # A3
```

```
    else:                                   # ch为应该排除的数：滑窗清空，从right+1开始下一个滑窗
        left = right+1
        hash_table = collections.Counter()

print(ans)
```

Java

```
import java.util.Scanner;

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

        int left = 0;
        int ans = 0;
        int[] hashTable = new int[128]; // Array to record the frequency of characters in the window

        for (int right = 0; right < s.length(); right++) {
            char ch = s.charAt(right);

            if (ch != a.charAt(0)) { // If ch is not the character to be excluded, slide the window
                hashTable[ch]++; // A1
                while (hashTable[ch] == 3) { // A2
                    char leftChar = s.charAt(left);
                    hashTable[leftChar]--;
                    left++;
                }
                ans = Math.max(ans, right - left + 1); // A3
            } else { // If ch is the character to be excluded, clear the window and start a new one from right+1
                left = right + 1;
                hashTable = new int[128];
            }
        }

        System.out.println(ans);
    }
}

```

C++

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

using namespace std;

int main() {
    string a;
    getline(cin, a);

    string s;
    getline(cin, s);

    int left = 0;
    int ans = 0;
    int hashTable[128] = {0}; // Array to record the frequency of characters in the window

    for (int right = 0; right < s.length(); right++) {
        char ch = s[right];

        if (ch != a[0]) { // If ch is not the character to be excluded, slide the window
            hashTable[ch]++; // A1
            while (hashTable[ch] == 3) { // A2
                char leftChar = s[left];
                hashTable[leftChar]--;
                left++;
            }
            ans = max(ans, right - left + 1); // A3
        } else { // If ch is the character to be excluded, clear the window and start a new one from right+1
            left = right + 1;
            fill(hashTable, hashTable + 128, 0);
        }
    }

    cout << ans << endl;

    return 0;
}

```

解法二

（对第二个限制条件的第二种处理方式）

Python

```python
# 题目：2024E-寻找符合要求的最长子串
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：不定滑窗
# 代码看不懂的地方，请直接在群上提问

import collections

a = input()
s = input()

ans = 0
lst = s.split(a)                          # 根据指定字符a对s进行切割，得到单词列表lst
for word in lst:                          # 对于列表中的每一个单词word分别进行滑窗过程
    left, temp = 0, 0
    hash_table = collections.Counter()    # 哈希表记录窗口中各个字符出现频数
    # 对word进行滑窗，word中最大值为temp
    for right, ch in enumerate(word):
        hash_table[ch] += 1               # A1
        while (hash_table[ch] == 3):      # A2
            hash_table[word[left]] -= 1
            left += 1
        temp = max(temp, right-left+1)    # A3
    ans = max(ans, temp)                  # 每次滑窗过程结束后的结果temp，都要拿来更新ans

print(ans)
```

Java

```java
import java.util.Scanner;

public class Main {
```

```java
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String a = scanner.nextLine();
        String s = scanner.nextLine();

        int ans = 0;
        String[] words = s.split(a);  // Split the string 's' based on 'a'

        for (String word : words) {
            int left = 0;
            int temp = 0;
            int[] hashTable = new int[128]; // An array to record the frequency of characters in the window

            for (int right = 0; right < word.length(); right++) {
                char ch = word.charAt(right);

                if (ch != a.charAt(0)) { // If 'ch' is not the character to be excluded, slide the window
                    hashTable[ch]++; // A1
                    while (hashTable[ch] == 3) { // A2
                        char leftChar = word.charAt(left);
                        hashTable[leftChar]--;
                        left++;
                    }
                    temp = Math.max(temp, right - left + 1); // A3
                } else { // If 'ch' is the character to be excluded, clear the window and start a new one from right+1
                    left = right + 1;
                    hashTable = new int[128];
                }
            }

            ans = Math.max(ans, temp);
        }

        System.out.println(ans);
    }
}

```

C++

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

```cpp
#include <vector>
#include <algorithm>
#include <unordered_map>

using namespace std;

int main() {
    string a;
    getline(cin, a);

    string s;
    getline(cin, s);

    int ans = 0;
    vector<string> words;
    size_t pos = 0;
    while ((pos = s.find(a)) != string::npos) {
        words.push_back(s.substr(0, pos));
        s.erase(0, pos + a.length());
    }
    words.push_back(s);

    for (const string& word : words) {
        int left = 0;
        int temp = 0;
        unordered_map<char, int> hashTable;

        for (int right = 0; right < word.length(); right++) {
            char ch = word[right];
            hashTable[ch]++; // A1
            while (hashTable[ch] == 3) { // A2
                char leftChar = word[left];
                hashTable[leftChar]--;
                left++;
            }
            temp = max(temp, right - left + 1); // A3
        }
        ans = max(ans, temp);
    }

    cout << ans << endl;

    return 0;
}

```

时空复杂度

时间复杂度： `O(N)` 。仅需一次遍历整个字符串 `s` 。

空间复杂度： `O(N)` 。哈希表所占用空间。