【华为OD机考 统一考试机试C卷】 最长的指定瑕疵度的元音子串（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

开头和结尾都是元音字母（aeiouAEIOU）的字符串为元音字符串，其中混杂的非元音字母数量为其瑕疵度。比如:

1. “a” 、 “aa”是元音字符串，其瑕疵度都为0
2. “aiur”不是元音字符串（结尾不是元音字符）
3. “abira”是元音字符串，其瑕疵度为2

给定一个字符串，请找出指定瑕疵度的最长元音字符子串，并输出其长度，如果找不到满足条件的元音字符子串，输出0。

子串：字符串中任意个连续的字符组成的子序列称为该字符串的子串。

输入描述

首行输入是一个整数，表示预期的瑕疵度flaw，取值范围[0, 65535]。

接下来一行是一个仅由字符a-z和A-Z组成的字符串，字符串长度(0, 65535]。

输出描述

输出为一个整数，代表满足条件的元音字符子串的长度。

用例

输入	0 asdbuiodevauufgh
输出	3
说明	无

C++

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

using namespace std;

int main() {
    int flaw;
    cin >> flaw;
    cin.ignore(); // 读取换行符
    string s;
    cin>>s;
    unordered_set<char> vowels = {'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'};
    vector<int> vowelIdxs;
    for (int i = 0; i < s.length(); i++) {
        if (vowels.count(s[i])) {
            vowelIdxs.push_back(i);
        }
    }
    int left = 0, right = 0 ;
    vector<int> lengths;
    while (right < vowelIdxs.size()) {
        int lengthDiff = vowelIdxs[right] - vowelIdxs[left] - (right - left);
        if (lengthDiff > flaw) {

```

```
            left++;
        } else {
            if (lengthDiff == flaw) {
                lengths.push_back(vowelIdxs[right] - vowelIdxs[left] + 1);
            }
            right++;
        }
    }
    if (lengths.empty()) {
        cout << 0 << endl;
        return 0;
    }
    sort(lengths.rbegin(), lengths.rend());
    cout << lengths[0] << endl;
    return 0;
}
```

java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // 输入瑕疵度
        int flaw = scanner.nextInt();
        scanner.nextLine(); // 读取换行符
        // 输入字符串
        String s = scanner.nextLine();
        // 定义元音字母集合
        Set<Character> vowels = new HashSet<>(Arrays.asList('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'));
        // 记录字符串中所有元音字母的下标
        List<Integer> vowelIdxs = new ArrayList<>();
        for (int i = 0; i < s.length(); i++) {
            if (vowels.contains(s.charAt(i))) {
                vowelIdxs.add(i);
            }
        }
        // 初始化双指针
```

```
        int left = 0, right = 0;
        // 记录所有满足瑕疵度的元音子串的长度
        List<Integer> lengths = new ArrayList<>();
        while (right < vowelIdxs.size()) {
            // 计算当前子串的瑕疵度
            int lengthDiff = vowelIdxs.get(right) - vowelIdxs.get(left) - (right - left);
            if (lengthDiff > flaw) {
                // 如果瑕疵度超过了预期，左指针右移
                left++;
            } else {
                // 如果瑕疵度不超过预期，记录子串长度
                if (lengthDiff == flaw) {
                    lengths.add(vowelIdxs.get(right) - vowelIdxs.get(left) + 1);
                }
                // 右指针右移
                right++;
            }
        }
        // 如果没有满足瑕疵度的元音子串，输出 0
        if (lengths.isEmpty()) {
            System.out.println(0);
            return;
        }
        // 输出最长的元音子串的长度
        Collections.sort(lengths, Collections.reverseOrder());
        System.out.println(lengths.get(0));
    }
}
```

javaScript

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

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

rl.on('line', (line) => {
  // 输入瑕疵度
```

```
const flaw = parseInt(line.trim());
// 输入字符串
rl.on('line', (line) => {
  const s = line.trim();
  // 定义元音字母集合
  const vowels = new Set(['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']);
  // 记录字符串中所有元音字母的下标
  const vowelIdxs = [];
  for (let i = 0; i < s.length; i++) {
    if (vowels.has(s.charAt(i))) {
      vowelIdxs.push(i);
    }
  }
  // 初始化双指针
  let left = 0, right = 0 ;
  // 记录所有满足瑕疵度的元音子串的长度
  const lengths = [];
  while (right < vowelIdxs.length) {
    // 计算当前子串的瑕疵度
    const lengthDiff = vowelIdxs[right] - vowelIdxs[left] - (right - left);
    if (lengthDiff > flaw) {
      // 如果瑕疵度超过了预期，左指针右移
      left++;
    } else {
      // 如果瑕疵度不超过预期，记录子串长度
      if (lengthDiff === flaw) {
        lengths.push(vowelIdxs[right] - vowelIdxs[left] + 1);
      }
      // 右指针右移
      right++;
    }
  }
  // 如果没有满足瑕疵度的元音子串，输出 0
  if (lengths.length === 0) {
    console.log(0);
    return;
  }
  // 输出最长的元音子串的长度
  lengths.sort((a, b) => b - a);
  console.log(lengths[0]);
```

```
  });
});
```

python

```python
from typing import List
vowels = set(['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'])

def findLongestVowelSubstring(flaw: int, s: str) -> int:
    vowelIdxs = [i for i in range(len(s)) if s[i] in vowels]
    left, right= 0, 0
    lengths = []
    while right < len(vowelIdxs):
        lengthDiff = vowelIdxs[right] - vowelIdxs[left] - (right - left)
        if lengthDiff > flaw:
            left += 1
        else:
            if lengthDiff == flaw:
                lengths.append(vowelIdxs[right] - vowelIdxs[left] + 1)
            right += 1
    if not lengths:
        return 0
    return max(lengths)

flaw = int(input())
s = input()
print(findLongestVowelSubstring(flaw, s))
```

C语言

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_SIZE 65536 // 定义最大字符串大小

// 检查字符是否是元音
int isVowel(char c) {
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||

```

```
            c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U');
}

int main() {
    char s[MAX_SIZE]; // 存储输入的字符串
    int flaw; // 存储预期的瑕疵度

    // 读取瑕疵度和字符串
    scanf("%d", &flaw);
    scanf("%s", s);

    int length = strlen(s); // 字符串长度
    int vowelIdxs[MAX_SIZE]; // 存储元音字母下标的数组
    int idxCount = 0; // 元音字母数量
    int maxLength = 0; // 最长元音字符子串的长度

    // 记录所有元音字母的下标
    for (int i = 0; i < length; i++) {
        if (isVowel(s[i])) {
            vowelIdxs[idxCount++] = i;
        }
    }

    // 使用双指针找出满足瑕疵度的最长元音字符子串
    for (int i = 0; i < idxCount; i++) {
        for (int j = i; j < idxCount; j++) {
            // 计算当前子串的瑕疵度
            int currentFlaw = vowelIdxs[j] - vowelIdxs[i] - (j - i);
            if (currentFlaw == flaw) {
                // 如果瑕疵度符合预期，更新最长子串长度
                int currentLength = vowelIdxs[j] - vowelIdxs[i] + 1;
                if (currentLength > maxLength) {
                    maxLength = currentLength;
                }
            }
        }
    }

    // 输出最长的元音子串的长度
    printf("%d\n", maxLength);

```

```
    return 0;
}
```

完整用例

用例1

```
0
asdbuiodevauufgh
```

用例2

```
2
aeueo
```

用例3

```
0
aeiou
```

用例4

```
0
abcde
```

用例5

```
3
aeioubcdfg
```

用例6

```
10
aeioubcdfg
```

用例7

```
0
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

用例8

```
3
asdbuiodevauufgh
```

用例9

```
0
asdbuiodevauufgh
```

用例10

```
5
abcaeioubcde
```

文章目录

用例5
用例6
用例7
用例8
用例9
用例10