华为OD机试 – 完美走位（Java & JS & Python）

原创 伏城之外 已于 2023-02-15 13:51:05 修改 9570 收藏 30 版权
分类专栏：华为OD机试（Java & JS & Python） # 华为OD机试2023A 文章标签：算法 JavaScript 华为机试 Java Python

题目描述

在第一人称射击游戏中，玩家通过键盘的A、S、D、W四个按键控制游戏人物分别向左、向后、向右、向前进行移动，从而完成走位。

假设玩家每按动一次键盘，游戏任务会向某个方向移动一步，如果玩家在操作一定次数的键盘并且各个方向的步数相同时，此时游戏任务必定会回到原点，则称此次走位为完美走位。

现给定玩家的走位（例如：ASDA），请通过更换其中**一段连续走位的方式**使得原走位能够变成一个完美走位。其中待更换的连续走位可以是相同长度的任何走位。

请返回待更换的连续走位的最小可能长度。

如果原走位本身是一个完美走位，则返回0。

输入描述

输入为由键盘字母表示的走位s，例如：ASDA

输出描述

输出为待更换的连续走位的最小可能长度。

用例

输入	WASDAASD
输出	1
说明	将第二个A替换为W，即可得到完美走位

输入	AAAA
输出	3
说明	将其中三个连续的A替换为WSD，即可得到完美走位

题目解析

题目要求，保持W,A,S,D字母个数平衡，即相等，如果不相等，可以从字符串中选取一段连续子串替换，来让字符串平衡。

比如：WWWWAAAASSSS

字符串长度12，W,A,S,D平衡的话，则每个字母个数应该是3个，而现在W,A,S各有4个，也就是说各超了1个。

因此我们应该从字符串中，选取一段包含1个W，1个A，1个S的子串，来替换为D。

WWWWAAAASSSS

WWWWAAAASSSS

WWWWAAAASSSS

....

WWWWAAAASSSS

而符合这种要求的子串可能很多，我们需要找出其中最短的，即WAAAAS。

本题其实就是求最小覆盖子串，同LeetCode - 76 最小覆盖子串_伏城之外的博客-CSDN博客

题目解析请看上面链接博客。

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,
});

rl.on("line", (line) => {
  console.log(getResult(line));
});

function getResult(str) {
  // 此时count记录统计W,A,S,D字母的数量
  const count = {
    W: 0,
    A: 0,
    S: 0,
    D: 0,
  };

  for (let c of str) count[c]++;

  // 平衡状态时，W,A,S,D应该都是avg数量
  const avg = str.length / 4;

  let total = 0; // total用于记录多余字母个数

  let flag = true; // flag表示当前是否为平衡状态，默认是
  for (let c in count) {
    if (count[c] > avg) {
      flag = false; // 如果有一个字母数量超标，则平衡打破
      count[c] -= avg; // 此时count记录每个字母超过avg的数量
      total += count[c];
    } else {
      delete count[c];
    }
  }

  if (flag) return 0; // 如果平衡，则输出0

  let i = 0;
  let j = 0;
  let minLen = str.length + 1;

  while (j < str.length) {
    const jc = str[j];

    if (count[jc]-- > 0) {
      total--;
    }

    while (total === 0) {
      minLen = Math.min(minLen, j - i + 1);

      const ic = str[i];
      if (count[ic]++ >= 0) {
        total++;
      }

      i++;
    }

    j++;
  }

  return minLen;
}
```

Java算法源码

```java
import java.util.HashMap;
import java.util.Scanner;

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

  public static int getResult(String str) {
    // count用于记录W,A,S,D字母的数量
    HashMap<Character, Integer> count = new HashMap<>();

    for (int i = 0; i < str.length(); i++) {
      Character c = str.charAt(i);
      count.put(c, count.getOrDefault(c, 0) + 1);
    }

    // 平衡状态时，W,A,S,D应该都是avg数量
    int avg = str.length() / 4;

    // total用于记录多余字母个数
    int total = 0;

    // flag表示当前是否为平衡状态，默认是
    boolean flag = true;

    for (Character c : count.keySet()) {
      if (count.get(c) > avg) {
        // 如果有一个字母数量超标，则平衡打破
        flag = false;
        // 此时count记录每个字母超过avg的数量
        count.put(c, count.get(c) - avg);
        total += count.get(c);
      } else {
        count.put(c, 0); // 此时count统计的其实是多余字母，如果没有超过avg,则表示没有
      }
    }

    // 如果平衡，则输出0
    if (flag) return 0;

    int i = 0;
    int j = 0;
    int minLen = str.length() + 1;

    while (j < str.length()) {
      Character jc = str.charAt(j);

      if (count.get(jc) > 0) {
        total--;
      }
      count.put(jc, count.get(jc) - 1);

      while (total == 0) {
        minLen = Math.min(minLen, j - i + 1);

        Character ic = str.charAt(i);
        if (count.get(ic) >= 0) {
          total++;
        }
        count.put(ic, count.get(ic) + 1);

        i++;
      }
      j++;
    }
    return minLen;
  }
}
```

Python算法源码

```python
# 输入获取
s = input()


# 算法入口
def getResult(s):
    # 此时count记录统计W,A,S,D字母的数量
    count = {
        "W": 0,
        "A": 0,
        "S": 0,
        "D": 0
    }

    for c in s:
        count[c] += 1

    avg = len(s) / 4  # 平衡状态时，W,A,S,D应该都是avg数量
    total = 0  # total用于记录多余字母个数
    flag = True  # flag表示当前是否为平衡状态，默认是

    for c in count.keys():
        if count[c] > avg:
            flag = False  # 如果有一个字母数量超标，则平衡打破
            count[c] -= avg  # 此时count记录每个字母超过avg的数量
            total += count[c]
        else:
            count[c] = 0

    if flag:
        return 0  # 如果平衡，则输出0

    i = 0
    j = 0
    minLen = len(s) - 1

    while j < len(s):
        jc = s[j]

        if count[jc] > 0:
            total -= 1
        count[jc] -= 1

        while total == 0:
            minLen = min(minLen, j - i + 1)

            ic = s[i]
            if count[ic] >= 0:
                total += 1
            count[ic] += 1

            i += 1

        j += 1

    return minLen


print(getResult(s))
```