华为OD机试 – 最多颜色的车辆（Java & JS & Python）

原创 伏城之外 已于 2023-03-31 23:36:41 修改 4917 收藏 13 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

在一个狭小的路口，每秒只能通过一辆车，假设车辆的颜色只有 3 种，找出 N 秒内经过的最多颜色的车辆数量。

三种颜色编号为0，1，2

输入描述

第一行输入的是通过的车辆颜色信息

[0,1,1,2] 代表4 秒钟通过的车辆颜色分别是 0 , 1 , 1 , 2

第二行输入的是统计时间窗，整型，单位为秒

输出描述

输出指定时间窗内经过的最多颜色的车辆数量。

用例

输入	0 1 2 1 3
输出	2
说明	在 3 秒时间窗内，每个颜色最多出现 2 次。例如：[1,2,1]

输入	0 1 2 1 2
输出	1
说明	在 2 秒时间窗内，每个颜色最多出现1 次。

题目解析

简单的滑动窗口应用。我们可以利用相邻两个滑窗的差异比较，来避免重复的计算。

比如：下图是用例1的滑窗（黄色部分）运动过程

第二个滑窗相较于第一个滑窗而言，失去了0，新增了1，因此我们不需要重新统计第二个滑窗内部各种颜色的数量，只需要在第一个滑窗的统计结果基础上，减少0颜色数量1个，增加1颜色数量1个即可。

2023.03.31 根据考友反馈，本题会出现不止3种颜色，可能出现多种颜色，因此代码需要解除颜色限制。

下面代码已更新。

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 arr = lines[0].split(" ");
    const n = parseInt(lines[1]);

    console.log(getResult(arr, n));

    lines.length = 0;
  }
});

function getResult(arr, n) {
  // count用于统计滑动窗口内各种颜色的数目
  const count = {};

  // 初始滑动窗口的左右边界，注意这里的右边界r是不包含了，为了方便后面进行slice
  let l = 0;
  let r = l + n;

  // 统计初始滑动窗口中各种颜色的数量
  arr.slice(l, r).forEach((c) => {
    if (!count[c]) count[c] = 0;
    count[c]++;
  });

  // 将初始滑动窗口内部最多颜色数量给max
  let max = Math.max.apply(null, Object.values(count));

  // 如果滑动窗口右边界未达到数组尾巴，就继续右移
  // 注意，初始滑窗的右边界r是不包含的，因此r可以直接当成下一个滑窗的右边界使用
  while (r < arr.length) {
    // 当滑动窗口右移后，新的滑动窗口相比移动前来看，新增了arr[r]，失去了arr[l]，注意此时
    const add = arr[r++];
    const remove = arr[l++];

    if (!count[add]) count[add] = 0;
    count[add]++;

    if (!count[remove]) count[remove] = 0;
    count[remove]--;

    // 只有新增数量的颜色可能突破最大值
    max = Math.max(max, count[add]);
  }

  return max;
}
```

Java算法源码

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

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

    Integer[] arr =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray
    int n = sc.nextInt();

    System.out.println(getResult(arr, n));
  }

  public static int getResult(Integer[] arr, int n) {
    // count用于统计滑动窗口内各种颜色的数目
    HashMap<Integer, Integer> count = new HashMap<>();

    // 初始滑动窗口的左右边界，注意这里的右边界r是不包含的
    int l = 0;
    int r = l + n;

    // 记录滑窗内部最多颜色数量
    int max = 0;

    // 统计初始滑动窗口中各种颜色的数量
    for (int i = l; i < Math.min(r, arr.length); i++) {
      Integer c = arr[i];
      count.put(c, count.getOrDefault(c, 0) + 1);
      max = Math.max(max, count.get(c));
    }

    // 如果滑动窗口右边界未达到数组尾巴，就继续右移
    // 注意，初始滑窗的右边界r是不包含的，因此r可以直接当成下一个滑窗的右边界使用
    while (r < arr.length) {
      // 当滑动窗口右移后，新的滑动窗口相比移动前来看，新增了arr[r]，失去了arr[l]，注意此
      Integer add = arr[r++];
      Integer remove = arr[l++];

      count.put(add, count.getOrDefault(add, 0) + 1);
      count.put(remove, count.getOrDefault(remove, 0) - 1);

      // 只有新增数量的颜色可能突破最大值
      max = Math.max(max, count.get(add));
    }

    return max;
  }
}
```

Python算法源码

```python
# 输入获取
arr = input().split()
n = int(input())


# 算法入口
def getResult(arr, n):
    count = {}

    l = 0
    r = l + n

    for c in arr[l:r]:
        if count.get(c) is None:
            count[c] = 0
        count[c] += 1

    maxV = max(count.values())

    while r < len(arr):
        add = arr[r]
        remove = arr[l]

        r += 1
        l += 1

        if count.get(add) is None:
            count[add] = 0
        count[add] += 1

        if count.get(remove) is None:
            count[remove] = 0
        count[remove] -= 1

        maxV = max(maxV, count[add])

    return maxV


# 算法调用
print(getResult(arr, n))
```