【华为OD机考 统一考试机试C卷】小朋友来自多少小区（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

幼儿园组织活动，老师布置了一个任务：

每个小朋友去了解与自己同一个小区的小朋友还有几个。

我们将这些数量汇总到数组 garden 中。

请根据这些小朋友给出的信息，计算班级小朋友至少来自几个小区？

输入描述

输入：garden[] = {2, 2, 3}

- garden 数组长度最大为 999
- 每个小区的小朋友数量最多 1000 人，也就是 garden[i] 的范围为 [0, 999]

输出描述

输出：7

用例

输入

```
2 2 3
```

输出

```
7
```

说明

> 第一个小朋友反馈有两个小朋友和自己同一小区，即此小区有3个小朋友。
> 第二个小朋友反馈有两个小朋友和自己同一小区，即此小区有3个小朋友。
> 这两个小朋友，可能是同一小区的，且此小区的小朋友只有3个人。
> 第三个小区反馈还有3个小朋友与自己同一小区，则这些小朋友只能是另外一个小区的。这个小区有4个小朋友。

解题思路

题目要求计算的是班级小朋友至少来自几个小区，但实际上根据上面的用例看：本题的输出其实是至少的小朋友数量

如果两个小朋友反馈的同小区人数相同，我们可以假设他们来自同一个小区，并且将他们的小区视为一个整体进行计算。这样，我们可以通过合并相同反馈的小朋友来减少总的小区数，从而得出至少有多少小朋友的估计。

具体来说，如果有多个小朋友反馈了相同的同小区人数，我们可以将他们分成若干组，每组包含 y+1 个小朋友（因为每个小朋友包括他自己在内的小区总人数是 y+1）。

如果小朋友的数量不是 y+1 的整数倍，那么最后一组将包含不足 y+1 的小朋友，但仍然被视为一个独立的小区。因此，我们可以通过向上取整 x / (y+1) 来计算至少的小区数，其中 x 是反馈相同人数 y 的小朋友数量。

这种方法的关键在于，我们尽可能地将反馈相同的小朋友合并为同一个小区，以最小化小区的数量。通过这种策略，我们可以得出一个保守估计，即至少有多少小朋友参与了活动。

1. 为了解释这个过程，我们可以使用一个更复杂的用例：

假设我们有以下的报告情况：

- 有8个小朋友报告说有2个其他孩子和他们同一个小区
- 有5个小朋友报告说有4个其他孩子和他们同一个小区
- 有2个小朋友报告说有6个其他孩子和他们同一个小区

我们的目标是计算至少有多少个小区。

对于第一种情况：

- 每个报告实际上代表 `y + 1 = 3` 个孩子。
- 有8个报告，所以我们有 `8 * 3 = 24` 个孩子。
- 这24个孩子至少来自 `ceil(8 / 3) = 3` 个小区，因为每3个孩子至少来自1个小区。

对于第二种情况：

- 每个报告实际上代表 `y + 1 = 5` 个孩子。
- 有5个报告，所以我们有 `5 * 5 = 25` 个孩子。
- 这25个孩子至少来自 `ceil(5 / 5) = 1` 个小区，因为每5个孩子至少来自1个小区。

对于第三种情况：

- 每个报告实际上代表 `y + 1 = 7` 个孩子。
- 有2个报告，所以我们有 `2 * 7 = 14` 个孩子。
- 这14个孩子至少来自 `ceil(2 / 7) = 1` 个小区，因为每7个孩子至少来自1个小区。

将三种情况相加，我们得到至少有 `3 + 1 + 1 = 5` 个小区。

C++

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

using namespace std;
int main() {
```

```cpp
    // 使用字符串流读取一行输入并按空格分割
    string line;
    getline(std::cin, line);
    istringstream iss(line);
    string child;
    // 创建一个vector用于存储每个小区的孩子数量
    vector<int> counts;
    // 初始化结果变量为0，用于存储最终的小区数量
    int result = 0;

    // 遍历输入的孩子数量
    while (iss >> child) {
        // 将字符串转换为整数表示孩子数量
        int children =stoi(child);
        // 确保counts向量的长度足够
        while (children >= counts.size()) {
            // 如果不够，则在counts向量末尾添加0
            counts.push_back(0);
        }
        // 在对应的索引位置增加孩子数量
        counts[children]++;
    }

    // 遍历counts向量
    for (size_t i = 0; i < counts.size(); i++) {
        // 如果当前索引位置的值大于0
        if (counts[i] > 0) {
            // 计算每个小区的实际大小（孩子数量加上自己）
            int districtSize = i + 1;
            // 使用ceil进行向上取整计算至少需要的小区数量，并累加到结果变量中
            result +=ceil(static_cast<double>(counts[i]) / districtSize) * districtSize;
        }
    }

    // 输出最终计算的小区数量
    cout << result <<endl;
    return 0;
}
```

Java

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        // 创建Scanner对象用于读取标准输入
        Scanner sc = new Scanner(System.in);
        // 读取一行输入并按空格分割
        String[] input = sc.nextLine().split(" ");
        // 创建一个ArrayList用于存储每个小区的孩子数量
        List<Integer> counts = new ArrayList<>();
        // 初始化结果变量为0，用于存储最终的小区数量
        int result = 0;

        // 遍历输入的孩子数量
        for (String child : input) {
            // 将字符串转换为整数表示孩子数量
            int children = Integer.parseInt(child);
            // 确保counts列表的长度足够
            while (children >= counts.size()) {
                // 如果不够，则在counts列表末尾添加0
                counts.add(0);
            }
            // 在对应的索引位置增加孩子数量
            counts.set(children, counts.get(children) + 1);
        }

        // 遍历counts列表
        for (int i = 0; i < counts.size(); i++) {
            // 如果当前索引位置的值大于0
            if (counts.get(i) > 0) {
                // 计算每个小区的实际大小（孩子数量加上自己）
                int districtSize = i + 1;
                // 使用Math.ceil进行向上取整计算至少需要的小区数量，并累加到结果变量中
                result += Math.ceil((double)counts.get(i) / districtSize) * districtSize;
            }
        }

        // 输出最终计算的小区数量
        System.out.println(result);

```

```
    }
}
```

javaScript

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

// 创建readline接口实例
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// 读取一行输入
rl.on('line', (line) => {
  // 按空格分割输入的字符串
  const input = line.split(' ');
  // 创建一个数组用于存储每个小区的孩子数量
  const counts = [];
  // 初始化结果变量为0，用于存储最终的小区数量
  let result = 0;

  // 遍历输入的孩子数量
  input.forEach((child) => {
    // 将字符串转换为整数表示孩子数量
    const children = parseInt(child, 10);
    // 确保counts数组的长度足够
    while (children >= counts.length) {
      // 如果不够，则在counts数组末尾添加0
      counts.push(0);
    }
    // 在对应的索引位置增加孩子数量
    counts[children]++;
  });

  // 遍历counts数组
  counts.forEach((count, i) => {
    // 如果当前索引位置的值大于0
    if (count > 0) {
      // 计算每个小区的实际大小（孩子数量加上自己）
```

```
      const districtSize = i + 1;
      // 使用Math.ceil进行向上取整计算至少需要的小区数量，并累加到结果变量中
      result += Math.ceil(count / districtSize) * districtSize;
    }
  });

  // 输出最终计算的小区数量
  console.log(result);
  // 关闭readline接口实例
  rl.close();
});
```

Python

```
# 读取一行输入并按空格分割
input_str = input().split(' ')
# 创建一个列表用于存储每个小区的孩子数量
counts = []
# 初始化结果变量为0，用于存储最终的小区数量
result = 0

# 遍历输入的孩子数量
for child in input_str:
    # 将字符串转换为整数表示孩子数量
    children = int(child)
    # 确保counts列表的长度足够
    while children >= len(counts):
        # 如果不够，则在counts列表末尾添加0
        counts.append(0)
    # 在对应的索引位置增加孩子数量
    counts[children] += 1

# 遍历counts列表
for i, count in enumerate(counts):
    # 如果当前索引位置的值大于0
    if count > 0:
        # 计算每个小区的实际大小（孩子数量加上自己）
        districtSize = i + 1
        # 使用ceil进行向上取整计算至少需要的小区数量，并累加到结果变量中
        result += -(-count // districtSize) * districtSize
```

```
27
28
# 输出最终计算的小区数量
29
print(result)
```

C语言

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

// 主函数
int main() {
    // 定义变量
    int garden[1000];  // 存储每个小区的孩子数量
    int counts[1000] = {0};  // 记录每个数量出现的次数
    int result = 0;  // 存储最终的小区数量
    int n = 0, i;  // n为数组garden的实际长度，i用于循环
    int children;  // 临时变量，存储输入的孩子数量

    // 读取输入并存入数组garden
    while (scanf("%d", &children) != EOF) {
        garden[n++] = children;
    }

    // 遍历garden数组，计算每个数量出现的次数
    for (i = 0; i < n; i++) {
        children = garden[i];

        counts[children]++;  // 增加对应数量的计数
    }

    // 遍历counts数组，计算至少需要的小区数量
    for (i = 0; i < 1000; i++) {
        if (counts[i] > 0) {
            // 计算每个小区的实际大小（孩子数量加上自己）
            int districtSize = i + 1;
            // 使用ceil进行向上取整计算至少需要的小区数量，并累加到结果变量中
            result += ((counts[i] + districtSize - 1) / districtSize) * districtSize;
        }
    }

```

```
    // 输出最终计算的小区数量
    printf("%d\n", result);
    return 0;
}
```

文章目录