

华为OD机试 - 食堂供餐 (Java & JS & Python)

原创 伏城之外 于 2023-06-02 16:24:55 修改 1134 收藏 4
分类专栏: 华为OD机试AB (Java & JS & Python) 文章标签: 算法 华为机试 Java JavaScript Python

版权



题目描述

某公司员工食堂以盒饭方式供餐。

为将员工取餐排队时间降低为0，食堂的供餐速度必须要足够快。

现在需要根据以往员工取餐的统计信息，计算出一个刚好能达成排队时间为0的最低供餐速度。即，食堂在每个单位时间内必须至少做出多少份盒饭才能满足要求。



输入描述

第1行为一个正整数N，表示食堂开餐时长。

- $1 \leq N \leq 1000$

第2行为一个正整数M，表示开餐前食堂已经准备好的盒饭份数。

- $P_1 \leq M \leq 1000$

第3行为N个正整数，用空格分隔，依次表示开餐时间内按时间顺序每个单位时间进入食堂取餐的人数 P_i 。

- $1 \leq i \leq N$
- $0 \leq P_i \leq 100$

输出描述

一个整数，能满足题目要求的最低供餐速度（每个单位时间需要做出多少份盒饭）。

备注

- 每人只取一份盒饭。
- 需要满足排队时间为0，必须保证取餐员工到达食堂时，食堂库存盒饭数量不少于本次来取餐的人数。
- 第一个单位时间来取餐的员工只能取开餐前食堂准备好的盒饭。
- 每个单位时间里制作的盒饭只能供应给后续单位时间来的取餐的员工。
- 食堂在每个单位时间里制作的盒饭数量是相同的。

用例

输入	3 14 10 4 5
输出	3
说明	<p>本样例中，总共有3批员工就餐，每批人数分别为10、4、5。</p> <p>开餐前食堂库存14份。 食堂每个单位时间至少要做出3份餐饭才能达成排队时间为0的目标。具体情况如下：</p> <ol style="list-style-type: none">第一个单位时间来的10位员工直接从库存取餐。取餐后库存剩余4份盒饭，加上第一个单位时间做出的3份，库存有7份。第二个单位时间来的4员工从库存的7份中取4份。取餐后库存剩余3份盒饭，加上第二个单位时间做出的3份，库存有6份。第三个单位时间来的员工从库存的6份中取5份，库存足够。 <p>如果食堂在单位时间只能做出2份餐饭，则情况如下：</p> <ol style="list-style-type: none">第一个单位时间来的10位员工直接从库存取餐。取餐后库存剩余4份盒饭，加上第一个单位时间做出的2份，库存有6份第二个单位时间来的4员工从库存的6份中取4份。取餐后库存剩余2份盒饭，加上第二个单位时间做出的2份，库存有4份。第三个单位时间来的员工需要取5份，但库存只有4份，库存不够。
↓	

题目解析

本题主要考察 [二分法](#)。

排队前的初始盒饭数为m，假设每单位时间新增盒饭数add个

对于第一批到来的食客 $p[0]$ ，由于题目已将说了 $m \geq p[0]$ ，因此盒饭数是足够的，剩余盒饭数 $m -= p[0]$

第二批食客到来前，新增了add个盒饭，因此盒饭数变为 $m += add$

第二批食客 $p[1]$ 到来后，需要检查 $m \geq p[1]$ 是否成立，如果成立，则当前add满足第一批食客要求，做到0等待。如果不满足，则add不是一个符合要求。

按此逻辑，只要add保证，所有批次的食客都能0等待，那么add就是一个符合要求的解。

这种符合要求的add有很多个，我们需要取其中最小的add作为题解。

我们可以思考下：

add的最小值可以取多少？

比如当初始盒饭数m超级大，即每个单位不用新增盒饭数，都可以满足所有批次食客0等待，那么add可以取0。而0就是add能取得最小值。

add的最大值可以取多少？



是否存在一个add，必然能满足所有批次食客0等待呢？答案是有的，即add取 $\max(p)$ ，这样必然能够保证每个批次的食客都0等待。

我们可以二分取0~ $\max(p)$ 之间的值mid，作为add去尝试发饭，如果：

- mid可以让所有批次的食客都能0等待，则mid就是一个可能解，但不一定是最优解，因此我们需要记录 $ans = mid$ ，并且继续尝试更小的mid，即让 $max = mid - 1$
- mid不可以让所有批次的食客0等待，则mid取小了，因此下一轮二分，我们应该让mid取大一点，即 $min = max + 1$

最后返回ans即可。

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         int n = sc.nextInt();
9         int m = sc.nextInt();
10
11        int[] p = new int[n];
12        for (int i = 0; i < n; i++) p[i] = sc.nextInt();
13
14        System.out.println(getResult(n, m, p));
15    }
16
17    public static long getResult(int n, int m, int[] p) {
18        int min = 0; // 每个单位时间最少增加盒饭数量
19        int max = Arrays.stream(p).max().orElse(0); // 每个单位时间最多增加盒饭数量
20
21        // 记录题解
22        int ans = 0;
23
24        // 二分
25        while (min <= max) {
26            int mid = (min + max) >> 1;
27
28            // 检查每个单位时间增加mid份盒饭，是否可以满足0等待
29            if (check(m, mid, p)) {
30                // 可以满足的话，则mid就是一个可能解
31                ans = mid;
32                // 继续查找更优解
33                max = mid - 1;
34            } else {
35                // 不满足的话，则说明mid取小了，下一轮应该取更大的mid
36                min = mid + 1;
37            }
38        }
39    }
40
41    private static boolean check(int m, int mid, int[] p) {
42        int sum = 0;
43        for (int i = 0; i < p.length; i++) {
44            sum += p[i];
45            if (sum >= mid) {
46                sum = 0;
47            }
48        }
49        return sum == 0;
50    }
51}
```

```
38     }
39
40     return ans;
41 }
42
43 public static boolean check(int m, int add, int[] p) {
44     m -= p[0]; // P1 ≤ M ≤ 1000, 因此这里 m - p[0] 必然大于等于 0
45
46     for (int i = 1; i < p.length; i++) {
47         m += add;
48         if (m >= p[i]) m -= p[i];
49         else return false;
50     }
51
52     return true;
53 }
54 }
```