

题目描述

小明有 n 块木板，第 i ($1 \leq i \leq n$) 块木板长度为 a_i 。
小明买了一块长度为 m 的木料，这块木料可以切割成任意块，拼接到已有的木板上，用来加长木板。
小明想让最短的模板尽量长。请问小明加长木板后，最短木板的长度可以为多少？

输入描述

输入的第一行包含两个正整数， n ($1 \leq n \leq 10^3$), m ($1 \leq m \leq 10^6$)， n 表示木板数， m 表示木板长度。
输入的第二行包含 n 个正整数， a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^6$)。

输出描述

输出的唯一一行包含一个正整数，表示加长木板后，最短木板的长度最大可以为多少？

用例

输入	5 3 4 5 3 5 5
输出	5
说明	给第1块木板长度增加1，给第3块木板长度增加2后，这5块木板长度变为[5,5,5,5,5]，最短的木板的长度最大为5。

输入	5 2 4 5 3 5 5
输出	4
说明	给第3块木板长度增加1后，这5块木板长度变为[4,5,4,5,5]，剩余木料的长度为1。此时剩余木料无论给哪块木板加长，最短木料的长度都为4。

题目解析

本题的题意是比较明确的，我的解题思路如下：

要想让最短的木板尽可能长，那么我们就需要不停地递进式补足最短板，比如用例输入有5个板：4 5 3 5 5，可用材料 $m=3$

最短的板长度是3，只有一个，那么我们就将他补足到4，此时消耗了一单位长度的材料， $m=2$

这样的话，只剩下两种长度的板4，5，

且4长度有两个，5长度有三个，最短板是长度4。

接下来我们应该尽量将最短板4长度的板补足到5长度，而刚好剩余材料 $m=2$ ，可以将所有4长度的板补足到5长度，此时所有板都是5长度，且材料耗尽。

我们还需要考虑一种特殊情况，那就是 m 还有值，但是只剩下一种长度的板，此时我们应该平分材料到每一个板，

假设只剩一种长度的板有 count 个，则平均分的话，每个板能分得 m / count 长度，这个值有可能是小数，我们举个例子：

5个一样长度 x 的板， $m = 13$ ，则 $13 / 5 = 2...3$ ，因此最短板长度就是 $x+2$ ，

再比如

5个一样长度 x 的板， $m = 15$ ，则 $15 / 5 = 3$ ，因此最短板长度就是 $x+3$ 。

本题有点贪心思维，即优先分配 m 的长度给最短板。

关于算法的时间复杂度，由于板子数量最多1000个，因此统计相同长度板子数量的时间复杂度很低。

然后 m 的长度达到了 10^6 ，这就比较大了，但是我们通过不断递进式累计最短板数量，最终不会受到 m 长度的影响，只会受到板子数量的影响。

因此下面整体复杂度是 $O(N)$ ，且 N 取值最多是1000。

JavaScript 算法源码

伏城之外 已关注

0

0

2

6

6

专栏目录

已订阅

题目解析

本题的题意是比较明确的，我的解题思路如下：

要想让最短的木板尽可能长，那么我们就不断地递进式补足最短板，比如用例输入有5个板：4 5 3 5 5，可用材料 $m=3$

最短的板长度是3，只有一个，那么我们就将他补足到4，此时消耗了一单位长度的材料， $m=2$

这样的话，只剩下两种长度的板4，5，

且4长度有两个，5长度有三个，最短板是长度4。

接下来我们应该尽量将最短板4长度的板补足到5长度，而刚好剩余材料 $m=2$ ，可以将所有4长度的板补足到5长度，此时所有板都是5长度，且材料耗尽。

我们还需要考虑一种特殊情况，那就是 m 还有值，但是只剩一种长度的板，此时我们应该平分材料到每一个板，

假设只剩一种长度的板有 count 个，则平均的话，每个板能分得 m / count 长度，这个值有可能是小数，我们举个例子：

5个一样长度 x 的板， $m = 13$ ，则 $13 / 5 = 2...3$ ，因此最短板长度就是 $x+2$ ，

再比如

5个一样长度 x 的板， $m = 15$ ，则 $13 / 5 = 3$ ，因此最短板长度就是 $x+3$ 。

本题有点贪心思维，即优先分配 m 的长度给最短板。

关于算法的时间复杂度，由于板子数量最多1000个，因此统计相同长度板子数量的时间复杂度很低。

然后 m 的长度达到了 10^6 ，这就比较大了，但是我们通过不断递进式累计最短板数量，最终不会受到 m 长度的影响，只会受到板子数量的影响。

因此下面整体复杂度是 $O(N)$ ，且 N 取值最多是1000。

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12 });
13
14 if (lines.length === 2) {
15   const [n, m] = lines[0].split(" ").map(Number);
16   const a = lines[1].split(" ").map(Number);
17   console.log(getResult(m, a));
18
19   lines.length = 0;
20 }
21
22 function getResult(m, a) {
23   // 统计每种长度板的数量，记录到count中，属性是板长度，属性值是板数量
24   const count = {};
25   for (let ai of a) {
26     count[ai] ? count[ai]++ : (count[ai] = 1);
27   }
28
29   // 将统计到的板，按板长度升序
30   const arr = [];
31   for (let ai in count) {
32     arr.push([ai - 0, count[ai]]);
33   }
34   arr.sort((a, b) => a[0] - b[0]);
35
36   // 只要还有剩余的m长度，就给他补到最短板上
37   while (m > 0) {
38     // 如果只有一种板长度，那么就尝试将m平均分配到各个板上
39     if (arr.length === 1) {
40       const [len, count] = arr[0];
41       return len + Math.floor(m / count);
42     }
43
44     // 如果有多种板长度
45     // min1是最短板
46     let min1 = arr.shift();
47     // min2是第二短板
48     let min2 = arr[0];
49
50     // diff是最短板和第二最短板的差距
51     let diff = min2[0] - min1[0];
52
53     // 将所有最短板补足到第二短板的长度，所需要总长度total
54     let total = diff * min1[1];
55
56     // 如果m的长度不够补足所有最短板，那么说明此时最短板的长度就是题解
57     if (total > m) {
```

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const [n, m] = lines[0].split(" ").map(Number);
15     const a = lines[1].split(" ").map(Number);
16     console.log(getResult(m, a));
17   }
18   lines.length = 0;
19 }
20 });
21
22 function getResult(m, a) {
23   // 统计每种长度板的数量，记录到count中，属性是板长度，属性值是板数量
24   const count = {};
25   for (let ai of a) {
26     count[ai] ? count[ai]++ : (count[ai] = 1);
27   }
28
29   // 将统计到的板，按板长度升序
30   const arr = [];
31   for (let ai in count) {
32     arr.push([ai - 0, count[ai]]);
33   }
34   arr.sort((a, b) => a[0] - b[0]);
35
36   // 只要还有剩余的m长度，就将其补到最短板
37   while (m > 0) {
38     // 如果只有一种板长度，那么就会尝试将m平均分配到各个板上
39     if (arr.length === 1) {
40       const [len, count] = arr[0];
41       return len + Math.floor(m / count);
42     }
43
44     // 如果有多种板长度
45     // min1是最短板
46     let min1 = arr.shift();
47     // min2是第二最短板
48     let min2 = arr[0];
49
50     // diff是最短板和第二最短板的差距
51     let diff = min2[0] - min1[0];
52
53     // 将所有最短板补足到第二最短板的长度，所需要总长度total
54     let total = diff * min1[1];
55
56     // 如果m的长度不够补足所有最短板，那么说明此时最短板的长度就是题解
57     if (total > m) {
58       return min1[0] + Math.floor(m / min1[1]);
59     }
60     // 如果m的长度刚好可以补足所有最短板，那么说明最短板可以全部升级到第二短板，且刚好用完m，因此第二短板的长度就是题解
61     else if (total === m) {
62       return min2[0];
63     }
64     // 如果m的长度足够长，能补足所有最短板到第二短板，还能有剩余，则将最短的数量加到第二短板的数量上，继续下轮循环
65     else {
66       m -= total;
67       min2[1] += min1[1];
68     }
69   }
70
71   return arr[0][0];
72 }
```

Java算法源码

```
1  import java.util.HashMap;
2  import java.util.PriorityQueue;
3  import java.util.Scanner;
4
5  public class Main {
6    public static void main(String[] args) {
7      Scanner sc = new Scanner(System.in);
8
9      int n = sc.nextInt();
10     int m = sc.nextInt();
11
12     int[] a = new int[n];
13     for (int i = 0; i < n; i++) {
14       a[i] = sc.nextInt();
15     }
16   }
```

Java算法源码

```
1 import java.util.HashMap;
2 import java.util.PriorityQueue;
3 import java.util.Scanner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         int n = sc.nextInt();
10        int m = sc.nextInt();
11
12        int[] a = new int[n];
13        for (int i = 0; i < n; i++) {
14            a[i] = sc.nextInt();
15        }
16
17        System.out.println(getResult(m, a));
18    }
19
20    public static int getResult(int m, int[] a) {
21        // 统计每种长度板的数量，记录到woods中，key是板长度，val是板数量
22        HashMap<Integer, Integer> woods = new HashMap<>();
23        for (Integer ai : a) {
24            if (woods.containsKey(ai)) {
25                Integer val = woods.get(ai);
26                woods.put(ai, ++val);
27            } else {
28                woods.put(ai, 1);
29            }
30        }
31
32        // 将统计到的板，按板长度排优先级，长度越短优先级越高，这里使用优先队列来实现优先级
33        PriorityQueue<Integer[]> pq = new PriorityQueue<>((b, c) -> b[0] - c[0]);
34        for (Integer wood : woods.keySet()) {
35            pq.offer(new Integer[] {wood, woods.get(wood)});
36        }
37
38        // 只要还有剩余的m长度，就给他补到最短板
39        while (m > 0) {
40            // 如果只有一种板长度，那么就尝试将m平均分配到各个板上
41            if (pq.size() == 1) {
42                Integer[] info = pq.poll();
43                int len = info[0];
44                int count = info[1];
45                return len + m / count;
46            }
47
48            // 如果有多种板长度
49            // min1是最短板
50            Integer[] min1 = pq.poll();
51            // min2是第二最短板
52            Integer[] min2 = pq.peek();
53
54            // diff是最短板和第二最短板的差距
55            int diff = min2[0] - min1[0];
56            // 将所有最短板补足到第二短板的长度，所需要总长度total
57            int total = diff * min1[1];
58
59            // 如果m的长度不够补足所有最短板，那么说明此时最短板的长度就是题解
60            if (total > m) {
61                return min1[0] + m / min1[1];
62            }
63            // 如果m的长度刚好可以补足所有最短板，那么说明最短板可以全部升级到第二短板，且刚好用完m，因此第二短板的长度就是题解
64            else if (total == m) {
65                return min2[0];
66            }
67            // 如果m的长度足够长，能补足所有最短板到第二短板，还能有剩余，则将最短的数量加到第二短板的数量上，继续下轮循环
68            else {
69                m -= total;
70                min2[1] += min1[1];
71            }
72        }
73
74        return pq.peek()[0];
75    }
76 }
```

Python算法源码

```
1 # 输入获取
2 import math
3
4 n, m = map(int, input().split())
5 a = list(map(int, input().split()))
6
7
8 # 算法入口
9 def getResult(m, a):
10     # 统计每种长度板的数量，记录到count中，属性是板长度，属性值是板数量
11     count = {}
```

Python算法源码

```
1 # 输入获取
2 import math
3
4 n, m = map(int, input().split())
5 a = list(map(int, input().split()))
6
7
8 # 算法入口
9 def getResult(m, a):
10     # 统计每种长度板的数量，记录到count中，属性是板长度，属性值是板数量
11     count = {}
12     for ai in a:
13         if count.get(ai) is None:
14             count[ai] = 1
15         else:
16             count[ai] += 1
17
18     # 将统计到的板，按板长度升序
19     arr = []
20     for ai in count.keys():
21         arr.append([int(ai), count[ai]])
22
23     arr.sort(key=lambda x: x[0])
24
25     # 只要还有剩余的m长度，就将其补到最短板上
26     while m > 0:
27         # 如果只有一种板长度，那么就会尝试将m平均分配到各个板上
28         if len(arr) == 1:
29             lenV, count = arr[0]
30             return lenV + math.floor(m / count)
31
32         # 如果有多种板长度
33         min1 = arr.pop(0) # min1是最短板
34         min2 = arr[0] # min2是第二最短板
35
36         # diff是最短板和第二最短板的差距
37         diff = min2[0] - min1[0]
38
39         # 将所有最短板补足到第二短板的长度，所需要总长度total
40         total = diff * min1[1]
41
42         # 如果m的长度不够补足所有最短板，那么说明此时最短板的长度就是题解
43         if total > m:
44             return min1[0] + math.floor(m / min1[1])
45         # 如果m的长度刚好可以补足所有最短板，那么说明最短板可以全部升级到第二短板，且刚好用完m，因此第二短板的长度就是题解
46         elif total == m:
47             return min2[0]
48         # 如果m的长度足够长，能补足所有最短板到第二短板，还能有剩余，则将最短的数量加到第二短板的数量上，继续下轮循环
49         else:
50             m -= total
51             min2[1] += min1[1]
52
53     return arr[0][0]
54
55
56 # 算法调用
57 print(getResult(m, a))
```