【二分查找】-孙悟空吃蟠桃

题目描述与示例

题目描述

孙悟空喜欢吃蟠桃，一天他趁守卫蟠桃园的天兵天将离开了而偷偷的来到王母娘娘的蟠桃园偷吃蟠桃。

已知蟠桃园有 `N` 棵蟠桃树，第 `i` 棵蟠桃树上有 `N[i]` （大于 `0` ）个蟠桃，天兵天将将在 `H` （不小于蟠桃树棵数）小时后回来。

孙悟空可以决定他吃蟠桃的速度 `K` （单位：个/小时），每个小时他会选择一颗蟠桃树，从中吃掉 `K` 个蟠桃，如果这棵树上的蟠桃数小于 `K` ，他将吃掉这棵树上所有蟠桃，然后这一小时内不再吃其余蟠桃树上的蟠桃。

孙悟空喜欢慢慢吃，但仍想在天兵天将回来前将所有蟠桃吃完。

求孙悟空可以在 `H` 小时内吃掉所有蟠桃的最小速度 `K` （ `K` 为整数）。

输入描述

第一行输入为 `N` 个数字， `N` 表示桃树的数量，这 `N` 个数字表示每颗桃树上蟠桃的数量

第二行输入为一个数字，表示守卫离开的时间 `H` 。

其中数字通过空格分割， `N` 、 `H` 为正整数，每颗树上都有蟠桃，且 `0 < N < 10000` ， `0 < H < 10000` 。

输出描述

吃掉所有蟠桃的最小速度 `K` （ `K` 为整数），无解或者输入异常时输出 `0` 。

示例一

输入

```
```

```
1 3 11 6 7 8
2 1
```

输出

```
1 0
```

示例二

输入

```
1 3 11 6 7 8
2 5
```

输出

```
1 11
```

解题思路

> 注意，本题和LeetCode875.爱吃香蕉的珂珂完全一致。直接按照课上的写法完成即可。唯一需要特殊判断的是，当 `nums` 数组长度大于 `h` 时，必然无法在h小时内吃完所有蟠桃，直接输出 `0`

代码

Python

```python
# 题目：【二分查找】2024E-孙悟空吃蟠桃
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：二分查找
# 代码看不懂的地方，请直接在群上提问
# 相关题目：LeetCode875.爱吃香蕉的珂珂


# 导入向上取整函数ceil，用于后续的计算
from math import ceil


nums = list(map(int, input().split()))
h = int(input())


# 计算在速度k的条件下，所花费的时间h的函数
def cal_hour_used(nums, k):
    return sum(ceil(p / k) for p in nums)


# 二分查找求解问题的函数
def minEatingSpeed(nums, h):
    # 左闭右开
    left, right = 1, max(nums) + 1
    while left < right:
        mid = (left + right) // 2
        # 花费时间太少，速度偏大，速度还可以减小，
        # 搜索区间向左折半，right可以向左移动
        if cal_hour_used(nums, mid) <= h:
            right = mid
        else:
            left = mid + 1
    return left


# 如果nums的长度已经大于h，一定无法在h小时内吃完所有蟠桃
# 直接输出0
if len(nums) > h:
    print(0)
# 否则进行二分，输出答案
else:
    print(minEatingSpeed(nums, h))
```

Java

```java
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] numsStr = scanner.nextLine().split(" ");
        int[] nums = new int[numsStr.length];
        for (int i = 0; i < numsStr.length; i++) {
            nums[i] = Integer.parseInt(numsStr[i]);
        }
        int h = scanner.nextInt();

        int left = 1;
        int right = getMax(nums) + 1;

        while (left < right) {
            int mid = left + (right - left) / 2;
            if (calHourUsed(nums, mid) <= h) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        if (nums.length > h) {
            System.out.println(0);
        } else {
            System.out.println(left);
        }
    }

    public static int calHourUsed(int[] nums, int k) {
        int hour = 0;
        for (int p : nums) {
            hour += Math.ceil((double) p / k);
        }
        return hour;
    }

    public static int getMax(int[] nums) {
        int max = Integer.MIN_VALUE;
        for (int num : nums) {
```

```
            max = Math.max(max, num);
        }
        return max;
    }
}

```

C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <climits>
using namespace std;

int calHourUsed(vector<int>& nums, int k) {
    int hour = 0;
    for (int p : nums) {
        hour += ceil((double) p / k);
    }
    return hour;
}

int getMax(vector<int>& nums) {
    int max = INT_MIN;
    for (int num : nums) {
        max = std::max(max, num);
    }
    return max;
}

int main() {
    string input;
    getline(cin, input);
    string input2;
    getline(cin, input2);

    vector<int> nums;
    size_t pos = 0;
    while ((pos = input.find(' ')) != string::npos) {
        nums.push_back(stoi(input.substr(0, pos)));
        input.erase(0, pos + 1);
    }
    nums.push_back(stoi(input));
```

```
37
38      int h = stoi(input2);
39
40      int left = 1;
41      int right = getMax(nums) + 1;
42
43      while (left < right) {
44          int mid = left + (right - left) / 2;
45          if (calHourUsed(nums, mid) <= h) {
46              right = mid;
47          } else {
48              left = mid + 1;
49          }
50      }
51
52      if (nums.size() > h) {
53          cout << 0 << endl;
54      } else {
55          cout << left << endl;
56      }
57
58      return 0;
59  }
60
```

时空复杂度

- 时间复杂度：`O(NlogN)`。其中`N`为`nums`数组长度。
- 空间复杂度：`O(1)`。