【华为OD机考 统一考试机试C卷】 机器人仓库搬砖（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

机器人搬砖，一共有N堆砖存放在N个不同的仓库中，第i堆砖中有bricks[i]块砖头，要求在8小时内搬完。机器人每小时能搬砖的数量取决于有多少能量格，机器人一个小时中只能在一个仓库中搬砖，机器人的能量格每小时补充一次且能量格只在这一个小时有效，为使得机器人损耗最小化尽量减小每次补充的能量格数 为了保障在8小时内能完成搬砖任务，请计算每小时给机器人充能的最小能量格数。

1、无需考虑机器人补充能量格的耗时，
2、无需考虑机器人搬砖的耗时;
3、机器人每小时补充能量格只在这一个小时中有效;

输入描述

第一行为一行数字，空格分隔

输出描述

机器人每小时最少需要充的能量格，若无法完成任务，输出 -1

示例1

输入	30 12 25 8 19
输出	15

示例2

输入	10 12 25 8 19 8 6 4 17 19 20 30
输出	-1

解题思路

1. 首先，检查砖块的数量是否大于8，如果大于8，则返回-1。在本例中，砖块数量为5，因此不返回-1。
2. 初始化左右边界 `left` 和 `right` 。 `left` 设置为1（因为至少需要每小时1块能量）， `right` 设置为数组中的最大值（即30）。
3. 进入 `while` 循环执行二分查找。这个循环将不断缩小搜索范围直到找到最小的每小时能量块数量。
4. 在每次循环中，计算中间值 `middle` ，然后计算使用 `middle` 作为每小时能量块数量时，完成搬运所有砖块需要的总时间 `total_time` 。
5. 如果 `total_time` 大于 `hours` ，说明 `middle` 太小，不能在规定时间内完成任务，因此将 `left` 设置为 `middle + 1` 。
6. 如果 `total_time` 小于等于 `hours` ，说明 `middle` 可能太大或刚好合适，因此将 `right` 设置为 `middle` 以尝试找到更小的值。
7. 当 `left` 和 `right` 相遇时，循环结束，此时 `left` 即为我们寻找的最小能量块数量。
8. 最后，再次计算使用 `left` 作为每小时能量块数量时，完成搬运所有砖块需要的总时间 `sum` 。如果 `sum` 大于 `hours` ，说明即使是 `left` 也不能在规定时间内完成任务，因此返回-1。
9. 如果 `sum` 小于等于 `hours` ，则 `left` 是可以在规定时间内完成任务的最小能量块数量，返回 `left` 。

现在，我们来分析给定的用例：

- 输入： `30 12 25 8 19`
- 输出： `15`

执行过程如下：

1. 初始化 `left = 1` 和 `right = 30`。
2. 进入循环，计算 `middle`。
3. 第一次循环：`middle = (1 + 30) / 2 = 15`，计算 `total_time` 为 `2 + 1 + 2 + 1 + 2 = 8`。因为 `total_time` 等于 `hours`，所以 `right` 更新为 `15`。
4. 第二次循环：`middle = (1 + 15) / 2 = 8`，计算 `total_time` 为 `4 + 2 + 4 + 1 + 3 = 14`。因为 `total_time` 大于 `hours`，所以 `left` 更新为 `9`。
5. 第三次循环：`middle = (9 + 15) / 2 = 12`，计算 `total_time` 为 `3 + 1 + 3 + 1 + 2 = 10`。因为 `total_time` 大于 `hours`，所以 `left` 更新为 `13`。
6. 第四次循环：`middle = (13 + 15) / 2 = 14`，计算 `total_time` 为 `3 + 1 + 2 + 1 + 2 = 9`。因为 `total_time` 大于 `hours`，所以 `left` 更新为 `15`。
7. 第五次循环：`middle = (15 + 15) / 2 = 15`，由于 `left` 已经等于 `right`，循环结束。
8. 最后，`left` 为 `15`，计算 `sum` 为 `2 + 1 + 2 + 1 + 2 = 8`，等于 `hours`，因此返回 `15`。

所以，对于给定的输入 `30 12 25 8 19`，输出应该是 `15`，这表明每小时至少需要充能15块能量格，才能在8小时内完成搬砖任务。

C++

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

int minEnergyBlocks(vector<int>& bricks, int hours) {
    if (bricks.size() > 8) { // 如果砖块数量大于8
        return -1; // 返回-1
    }
    int left = 1, right = *max_element(bricks.begin(), bricks.end()); // 初始化左右边界
    while (left < right) { // 二分查找
        int middle = (left + right) / 2; // 取中间值
        int total_time = 0; // 计算当前能量块数量下能够搬完所有砖的总时间
        for (int i = 0; i < bricks.size(); i++) {
            total_time += ceil((double)bricks[i] / middle);
        }
        if (total_time > hours) { // 如果当前能量块数量下搬完所有砖的总时间超过了限定时间，缩小搜索范围
```

```
            left = middle + 1;
        } else { // 否则，减小能量块数量
            right = middle;
        }
    }
    int sum = 0;
    for (int i = 0; i < bricks.size(); i++) {
        sum += ceil((double)bricks[i] / left);
    }
    if (sum > hours) { // 检查最终确定的能量块数量是否能在规定时间内搬完所有砖
        return -1; // 无法在规定时间内搬完所有砖
    }
    return left; // 返回最小能量块数量
}

int main() {
    vector<int> bricks;
    int brick;
    while (cin >> brick) {
        bricks.push_back(brick);
    }
    cout << minEnergyBlocks(bricks, 8); // 调用函数并输出结果
    return 0;
}
```

Java

```
import java.util.*;

public class Main {
    public static int minEnergyBlocks(int[] bricks, int hours) {
        if (bricks.length > 8) { // 如果砖块数量大于8
            return -1; // 返回-1
        }
        int left = 1, right = Arrays.stream(bricks).max().getAsInt(); // 初始化左右边界
        while (left < right) { // 二分查找
            int middle = (left + right) / 2; // 取中间值
            int total_time = 0; // 计算当前能量块数量下能够搬完所有砖的总时间
            for (int i = 0; i < bricks.length; i++) {
                total_time += Math.ceil((double) bricks[i] / middle);
```

```
            }
            if (total_time > hours) { // 如果当前能量块数量下搬完所有砖的总时间超过了限定时间，缩小搜索范围
                left = middle + 1;
            } else { // 否则，减小能量块数量
                right = middle;
            }
        }
        int sum = 0;
        for (int i = 0; i < bricks.length; i++) {
            sum += Math.ceil((double) bricks[i] / left);
        }
        if (sum > hours) { // 检查最终确定的能量块数量是否能在规定时间内搬完所有砖
            return -1; // 无法在规定时间内搬完所有砖
        }
        return left; // 返回最小能量块数量
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] input = scanner.nextLine().split(" ");
        int[] bricks = new int[input.length];
        for (int i = 0; i < input.length; i++) {
            bricks[i] = Integer.parseInt(input[i]);
        }
        System.out.println(minEnergyBlocks(bricks, 8)); // 调用函数并输出结果
    }
}
```

javaScript

```
function minEnergyBlocks(bricks, hours) {
    if (bricks.length > 8) { // 如果砖块数量大于8
        return -1; // 返回-1
    }
    let left = 1, right = Math.max(...bricks); // 初始化左右边界
    while (left < right) { // 二分查找
        let middle = Math.floor((left + right) / 2); // 取中间值
        let total_time = 0; // 计算当前能量块数量下能够搬完所有砖的总时间
        for (let i = 0; i < bricks.length; i++) {
            total_time += Math.ceil(bricks[i] / middle);
```

```
        }
        if (total_time > hours) { // 如果当前能量块数量下搬完所有砖的总时间超过了限定时间，缩小搜索范围
            left = middle + 1;
        } else { // 否则，减小能量块数量
            right = middle;
        }
    }
    let sum = 0;
    for (let i = 0; i < bricks.length; i++) {
        sum += Math.ceil(bricks[i] / left);
    }
    if (sum > hours) { // 检查最终确定的能量块数量是否能在规定时间内搬完所有砖
        return -1; // 无法在规定时间内搬完所有砖
    }
    return left; // 返回最小能量块数量
}

const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.on('line', (input) => {
    const bricks = input.split(' ').map(Number);
    console.log(minEnergyBlocks(bricks, 8)); // 调用函数并输出结果
    rl.close();
});
```

Python

```
import math

def min_energy_blocks(bricks, hours):
    if len(bricks) > 8:
        return -1
    # 初始化左右边界
    left, right = 1, max(bricks)
    # 二分查找
    while left < right:
```

```
        # 取中间值
        middle = (left + right) // 2
        # 计算当前能量块数量下能够搬完所有砖的总时间
        total_time = sum(math.ceil(i/middle) for i in bricks)
        if total_time > hours:
            # 如果当前能量块数量下搬完所有砖的总时间超过了限定时间，缩小搜索范围
            left = middle + 1
        else:
            # 否则，减小能量块数量
            right = middle
    # 检查最终确定的能量块数量是否能在规定时间内搬完所有砖
    if sum(math.ceil(i/left) for i in bricks) > hours:
        return -1  # 无法在规定时间内搬完所有砖
    return left  # 返回最小能量块数量

# 从输入中获取砖块数量列表
bricks = list(map(int, input().split()))
# 调用函数并输出结果
print(min_energy_blocks(bricks, 8))
```

C语言

```
#include <stdio.h>
#include <math.h>

#define MAX_BRICKS 100 // 定义最大砖块堆数

// 函数：计算机器人每小时最少需要充的能量格
int minEnergyBlocks(int bricks[], int n, int hours) {
    if (n > hours) { // 如果砖块数量大于时间限制
        return -1; // 无法完成，返回-1
    }

    // 寻找数组中最大元素
    int maxBrick = bricks[0];
    for (int i = 1; i < n; i++) {
        if (bricks[i] > maxBrick) {
            maxBrick = bricks[i];
        }
    }
```

```
    int left = 1, right = maxBrick; // 初始化左右边界
    while (left < right) { // 二分查找
        int middle = (left + right) / 2; // 取中间值
        int total_time = 0; // 初始化总时间
        // 计算在当前能量块数量下，完成所有砖块的总时间
        for (int i = 0; i < n; i++) {
            total_time += ceil((double)bricks[i] / middle);
        }

        // 判断是否能在规定时间内完成
        if (total_time > hours) {
            left = middle + 1;
        } else {
            right = middle;
        }
    }

    // 检查是否可以在规定时间内完成任务
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += ceil((double)bricks[i] / left);
    }
    if (sum > hours) {
        return -1; // 无法完成
    }
    return left; // 返回最小能量格数
}

int main() {
    int bricks[MAX_BRICKS]; // 定义砖块数组
    int n = 0; // 数组中元素的数量
    int brick;

    // 从标准输入读取砖块数量
    while (scanf("%d", &brick) != EOF) {
        bricks[n] = brick;
        n++;
    }
```

```
    // 调用函数并输出结果
    printf("%d\n", minEnergyBlocks(bricks, n, 8));
    return 0;
}
```

完整用例

用例1

5 5 5 5 5 5 5 5

用例2

1 1 1 1 1 1 1 100

用例3

10 10 10

用例4

10 20 30 40 50 60 70 80

用例5

8 8 8 8 8 8 8 8

用例6

150 120 150 120 150 150 120 150

用例7

45 30 60 20 55 35 50 40

用例8

7 13 19 25 31 37 43 49

用例9

12 23 34 45 56 67 78 89 90

用例10

30 12 25 8 19

文章目录


机考真题
华为OD
No No No
CSDN @算法大师