【华为OD机考 统一考试机试C卷】单行道汽车通行时间（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷） 。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多 ，按照之前的经验C卷D卷部分考题会复用A卷/B卷题，博主正积极从考过的同学收集C卷和D卷真题，可以查看下面的真题目录。

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 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为OD机考C卷和D卷真题华为OD机考C卷和D卷题解

题目描述

M（1<=M<=20）辆车需要在一条不能超车的单行道到达终点，起点到终点的距离为N（1<=N<=400）。速度快的车追上前车后，只能以前车的速度继续行驶。求最后一车辆到达目的地花费的时间。

注：每辆车固定间隔一小时出发，比如第一辆车0时出发，第二辆车1时出发，以此类推

输入描述

第一行两个数字：M N分别代表车辆数和到终点的距离，以空格分隔。

接下来M行，每行1个数字S，代表每辆车的速度。0<S<30。

输出描述

最后一辆车到达目的地花费的时间

用例

输入

```
2 11
3
```

输出

```
5.5
```

解题思路

参考代码注释

C++

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

int main() {
    // 获取车辆数M和终点距离N
    int M, N;
    cin >> M >> N;

    // 获取每辆车的速度并存储在vector speeds中
    vector<int> speeds(M);
    for (int index = 0; index < M; index++) {
        cin >> speeds[index];
    }

    // 初始化arrivalTimes vector，其中存储第一辆车到达目的地的时间
    vector<double> arrivalTimes(M);
    arrivalTimes[0] = (double) N / speeds[0];

    // 对于剩余的车辆，循环计算每辆车到达目的地的时间
    // 如果当前车辆比前一辆车更晚到达或与前一辆车同时到达，则更新时间并添加到arrivalTimes
    for (int index = 1; index < M; index++) {
        // 计算第index辆车单独行驶到目的地的时间，即终点距离N除以车速speeds[index]
```

```
        // 由于车辆是依次出发的，所以还需要加上车辆的出发延迟时间index
        double estimatedTime = (double) N / speeds[index] + index;

        // 比较当前车辆计算出的到达时间estimatedTime和前一辆车的到达时间arrivalTimes[index - 1]
        // 使用max函数确保当前车辆的到达时间不会早于前一辆车
        double adjustedTime = max(estimatedTime, arrivalTimes[index - 1]);
        arrivalTimes[index] = adjustedTime;
    }

    // 输出最后一辆车到达目的地的时间，但减去M再加1（这是因为车辆从0开始计数，而时间是从1开始计数）
    cout << arrivalTimes[M - 1] - M + 1 << endl;

    return 0;
}
```

Java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] inputLine = scanner.nextLine().split(" ");
        // 获取车辆数M和终点距离N
        int M = Integer.parseInt(inputLine[0]);
        int N = Integer.parseInt(inputLine[1]);

        // 获取每辆车的速度并存储在数组speeds中
        int[] speeds = new int[M];
        for (int index = 0; index < M; index++) {
            speeds[index] = Integer.parseInt(scanner.nextLine());
        }

        // 初始化arrivalTimes数组，其中存储第一辆车到达目的地的时间
        double[] arrivalTimes = new double[M];
        arrivalTimes[0] = (double) N / speeds[0];

        // 对于剩余的车辆，循环计算每辆车到达目的地的时间
        // 如果当前车辆比前一辆车更晚到达或与前一辆车同时到达，则更新时间并添加到arrivalTimes
        for (int index = 1; index < M; index++) {
```

```
            // 计算第index辆车单独行驶到目的地的时间，即终点距离N除以车速speeds[index]
            // 由于车辆是依次出发的，所以还需要加上车辆的出发延迟时间index
            double estimatedTime = (double) N / speeds[index] + index;

            // 比较当前车辆计算出的到达时间estimatedTime和前一辆车的到达时间arrivalTimes[index - 1]
            // 使用Math.max函数确保当前车辆的到达时间不会早于前一辆车
            double adjustedTime = Math.max(estimatedTime, arrivalTimes[index - 1]);
            arrivalTimes[index] = adjustedTime;
        }

        // 输出最后一辆车到达目的地的时间，但减去M再加1（这是因为车辆从0开始计数，而时间是从1开始计数）
        System.out.println(arrivalTimes[M - 1] - M + 1);
    }
}
```

javaScript

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

let lines = [];
rl.on('line', (line) => {
  lines.push(line);
}).on('close', () => {
  // 解析输入数据
  const [M, N] = lines[0].split(" ").map(Number);

  // 获取每辆车的速度并存储在数组speeds中
  const speeds = lines.slice(1, M + 1).map(Number);

  // 初始化arrivalTimes数组，其中存储第一辆车到达目的地的时间
  const arrivalTimes = new Array(M);
  arrivalTimes[0] = N / speeds[0];

  // 对于剩余的车辆，循环计算每辆车到达目的地的时间
  for (let index = 1; index < M; index++) {
    // 计算第index辆车单独行驶到目的地的时间
```

```
    const estimatedTime = N / speeds[index] + index;

    // 比较当前车辆计算出的到达时间和前一辆车的到达时间
    const adjustedTime = Math.max(estimatedTime, arrivalTimes[index - 1]);
    arrivalTimes[index] = adjustedTime;
  }

  // 输出最后一辆车到达目的地的时间，减去M再加1
  console.log(arrivalTimes[M - 1] - M + 1);
});
```

Python

```
# 导入必要的库
import sys

def main():
    # 读取输入：车辆数M和终点距离N
    M, N = map(int, input().split())

    # 获取每辆车的速度并存储在列表speeds中
    speeds = [int(input()) for _ in range(M)]

    # 初始化arrivalTimes列表，其中存储第一辆车到达目的地的时间
    arrivalTimes = [0] * M
    arrivalTimes[0] = N / speeds[0]

    # 对于剩余的车辆，计算每辆车到达目的地的时间
    for index in range(1, M):
        # 计算第index辆车单独行驶到目的地的时间
        estimatedTime = N / speeds[index] + index

        # 比较当前车辆计算出的到达时间和前一辆车的到达时间
        adjustedTime = max(estimatedTime, arrivalTimes[index - 1])
        arrivalTimes[index] = adjustedTime

    # 输出最后一辆车到达目的地的时间，减去M再加1
    print(arrivalTimes[M - 1] - M + 1)
```

```python
if __name__ == "__main__":
    main()
```

C语言

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

int main() {
    int M, N;

    // 读取输入：车辆数M和终点距离N
    scanf("%d %d", &M, &N);

    // 获取每辆车的速度并存储在数组speeds中
    int speeds[M];
    for (int index = 0; index < M; index++) {
        scanf("%d", &speeds[index]);
    }

    // 初始化arrivalTimes数组，其中存储第一辆车到达目的地的时间
    double arrivalTimes[M];
    arrivalTimes[0] = (double) N / speeds[0];

    // 对于剩余的车辆，计算每辆车到达目的地的时间
    for (int index = 1; index < M; index++) {
        // 计算第index辆车单独行驶到目的地的时间
        double estimatedTime = (double) N / speeds[index] + index;

        // 比较当前车辆计算出的到达时间和前一辆车的到达时间
        double adjustedTime = estimatedTime > arrivalTimes[index - 1] ? estimatedTime : arrivalTimes[index - 1];
        arrivalTimes[index] = adjustedTime;
    }

    // 输出最后一辆车到达目的地的时间，减去M再加1
    printf("%.1f\n", arrivalTimes[M - 1] - M + 1);

    return 0;
}
```

文章目录