讲评课

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.02

启航舱
中小学拔尖创新人才培养中心

目录

01

A

TEACHING
COURSEWARE

TEACH

题目描述

说明

判断一个正整数是否是两位数(即大于等于10且小于等于99)。若该正整数是两位数，输出1，否则输出0。

输入格式

一个正整数，不超过1000。

输出格式

一行。若该正整数是两位数，输出1，否则输出0。

样例

输入数据 1

```
54
```

Copy

输出数据 1

```
1
```

Copy

解题思路

思路1：
直接判断是否在10到99之间

思路2：
通过拆位计算是几位数

拆位

```cpp
#include<bits/stdc++.h>
using namespace std;
int f(int n)
{
    if(n == 0) return 1;
    int t = n;
    int ans = 0;
    while(t)
    {
        ans++;
        t/=10;
    }
    return ans;
}
int main()
{
    int n;
    cin>>n;
    if(f(n) == 2)
    {
        cout<<1<<endl;
    }
    else cout<<0<<endl;
}
```

02

B

TEACHING
COURSEWARE

TEACH

第一个X

第一个 X

输入文件 firstx.in 输出文件 firstx.out
时间限制 1000ms 空间限制 128MB

题目描述

现在 Alice 班里总共有 N 个人。他们在操场上坐成了一个圈。
每个人手里拿了一个大写字母 O 或者大写字母 X。
Alice 想知道从第 k 个人向后数，最少数几个数字，数到的人手里是一个大写字母 X。

注：由于形成了一个环，因此第 n 个人的下一个人是第 1 个人，同样的，第 1 个人前面的人是第 n 个人。

输入格式

第一行两个正整数 n 和 k，以空格分开。第一个数字表示 Alice 班里总共有 n 个人，第二个数字表示将要从第 k 个人开始向后数。
接下来一行一个字符串，共 n 个字母，每个字母是大写字母 O 和 X 中的一个。其中，第 i 个字母表示第 i 个人手上的字母。

输出格式

一个整数，表示向后数几个数后数到的人手里是大写字母 X。
保证至少有一个人手里是 X。

样例 #1

样例输入 #1

```
5 1
XXXXX
```

样例输出 #1

```
1
```

第一个X

样例 1 解释

最少向后数一个，第二个人手里有一个字母 X。

样例 #2

样例输入 #2

```
5 3
XOXOO
```

样例输出 #2

```
3
```

样例 2 解释

从第三个人开始，向后数三个（回到第一个人），他手里有一个字母 X。

大样例见文件 firstx.in/out

数据范围

共 10 个测试点，每个测试点 10 分。

测试点编号	数据范围	其他说明
#1~#5	$1 \le n,k \le 50$	无
#6	$1 \le n,k \le 1{,}000$	只有一个人手里有 X
#7~#10		无

第一个X

题目分析
这道题是说，Alice 班里的人围成一个圈，每个人拿的是字母 O 或者 X ，要从第 k 个人开始往后数，找出最少数几个数能碰到手里是 X 的人。因为是环形，所以数到最后会接着从头继续。那我们的任务就是模拟这个环形计数的过程，找到第一个 X 出现的位置与起始位置的间隔。
解题思路
首先，得处理环形的问题。比如总共有 n 个人，从第 k 个人开始数（这里要注意，题目里的第 k 个人，在数组里可能索引是 k - 1 ，因为数组索引从 0 开始 ）。
然后，从起始位置开始，依次往后遍历（利用取模运算来实现环形，比如当前索引是 i ，下一个就是（i + 1）% n ），每数一个人就检查是不是 X ，一旦碰到 X ，就计算数的步数，这个步数就是答案。

把环变成链：
在原数组后面加一个数组。

第一个X


```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    int n, k;
    string s;
    cin >> n >> k >> s;
    // 断环为链：将字符串拼接一份，形成长度为2n的线性结构
    string ss = s + s;
    // 起始位置（第k个人对应索引k-1）
    int start = k - 1;
    // 从起始位置开始查找，最多检查n个字符（覆盖整个环）
    for (int i = 0; i < n; ++i) {
        // 当前检查的位置是start + i
        if (ss[start + i] == 'X') {
            // 找到X，输出步数（i就是与起始位置的距离）
            cout << i << endl;
            return 0;
        }
    }
    // 题目保证至少有一个X，所以不会执行到这里
    return 0;
}
```

03

C

TEACHING
COURSEWARE

TEACH

咖啡

在花花大学，学生需要按照顺序去 n 间教室上课。第 i 间教室有一个属性 $s_i \in \{0, 1\}$。若 $s_i = 1$，表示这间教室有一座咖啡机；若 $s_i = 0$，表示这间教室没有咖啡机。

你是花花大学的一名学生。在一间有咖啡机的教室里，你可以通过饮用一杯咖啡来使得自己不会犯困。特别地，在你离开一间有咖啡机的教室后，你可以携带最多两杯咖啡（每只手可以拿一杯）前往下一间教室，这样即使那间教室没有咖啡机，你可以通过饮用你携带的咖啡来提神。

现在你想要知道，你最多可以在多少间教室内饮用咖啡。

输入格式

输入的第一行包含一个整数 n。

接下来一行，包含一个长度为 n 的，只由字符 `0` 与字符 `1` 构成的字符串 s，其中第 i 个字符 s_i 描述了第 i 间教室是否有咖啡机。

输出格式

输出一行一个整数，表示答案。

咖啡

遍历循环里，根据当前教室是否有咖啡机（s[i] 的值 ），分别处理喝咖啡和使用携带咖啡的情况，更新 ans 和 cnt 。比如遇到有咖啡机的，就增加 ans 并重置 cnt 为 2 ；遇到没有的，若有 cnt 就用掉来增加 ans 并减少 cnt 。

咖啡


```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    int n;
    string s;
    cin >> n >> s;  // 输入教室数量n和表示教室咖啡机情况的字符串s
    int ans = 0;  // 用于统计最多能喝咖啡的教室数量（原res）
    int cnt = 0;  // 记录当前能携带的咖啡数量（原carry）
    for (int i = 0; i < n; ++i) {  // 遍历每一间教室
        if (s[i] == '1') {  // 当前教室有咖啡机的情况
            ans++;  // 在这间教室喝一杯咖啡
            cnt = 2;  // 喝完后可以携带最多两杯咖啡前往下一间
        } else {  // 当前教室没有咖啡机的情况
            if (cnt > 0) {  // 如果有携带的咖啡
                ans++;  // 饮用携带的咖啡
                cnt--;  // 携带的咖啡数量减1
            }
        }
    }
    cout << ans << endl;  // 输出最多能喝咖啡的教室数量
    return 0;
}
```

04

D

TEACHING
COURSEWARE

TEACH

D

题目描述

小安经营着一家柠檬水摊位，一共营业 n 天。每天开始时，摊位会自动补充1个柠檬（可累积到后续天数使用）。在第 i 天，制作1杯柠檬水需要消耗 c_i 个柠檬（每天可以制作任意多杯）。请问小安在这 n 天里，最多能制作多少杯柠檬水？

输入格式

- 第一行：正整数 n（$1 \leq n \leq 2 \times 10^5$），表示营业的天数。
- 第二行：n 个正整数 $c_1, c_2, ..., c_n$（$1 \leq c_i \leq 10^9$），其中 c_i 表示第 i 天制作1杯柠檬水所需的柠檬数量。

输出格式

一行，一个非负整数，表示最多 n 天内最多能制作的柠檬水总杯数。

样例

输入：

```
6
3 2 5 3 4 3
```

Copy

输出：

```
2
```

Copy

D

样例说明

- 第1天开始时获得1个柠檬，累计1个（不足制作1杯，不制作）。
- 第2天开始时获得1个柠檬，累计2个，当天制作1杯（消耗2个），累计剩余0个。
- 第3天开始时获得1个柠檬，累计1个（不足制作）。
- 第4天开始时获得1个柠檬，累计2个（不足制作）。
- 第5天开始时获得1个柠檬，累计3个（不足制作）。
- 第6天开始时获得1个柠檬，累计4个，当天制作1杯（消耗3个），累计剩余1个。
- 总制作数量为2杯，无法再多制作。

数据范围

- 40% 的数据，满足 $1 \leq n \leq 100$，$1 \leq c_i \leq 100$。
- 100% 的数据，满足 $1 \leq n \leq 2 \times 10^5$，$1 \leq c_i \leq 10^9$。

D

1. 采用贪心算法，优先选择柠檬制作柠檬消耗最少的天数制作柠檬水，以最大化总制作杯数。因为在消耗少的天数制作，相同数量的柠檬能产出更多杯柠檬水，这是局部最优选择，最终能得到全局最优解。

2. 标记最优制作天数

3. 从后往前遍历每天的柠檬消耗，标记出消耗非递增的天数（即当前天数的消耗≤后面所有天数的消耗）。这些标记的天数是最优制作时机，因为它们在各自前面的天数中保持着最低消耗。

D

```cpp
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
const int N = 2e5 + 50;  // 最大天数
int cost[N];             // 每天制作1杯柠檬水所需的柠檬数量
bool is_best[N];         // 标记第i天是否为最优制作时机
int n;                   // 总天数
int main() {
    cin >> n;
    // 读取每天制作1杯柠檬水的柠檬消耗（1-based索引）
    for (int i = 1; i <= n; i++) {
        cin >> cost[i];
    }
    // 从后往前遍历，标记最优制作天数
    int min_cost = 1e9;  // 记录当前位置往后的最小消耗
    for (int i = n; i >= 1; i--) {
        // 若当前天数消耗≤ 后面所有天数的最小消耗，标记为最优制作时机
        if (cost[i] <= min_cost) {
            min_cost = cost[i];  // 更新最小消耗
            is_best[i] = true;   // 标记当前天为最优制作日
        }
    }
```

D


```
    int lemons = 0;          // 累计可用柠檬数（每天新增1个，间隔天数即积累数量）
    int last_day = 0;        // 上一次制作的天数
    int total_cups = 0;      // 总共可制作的柠檬水杯数
    // 从前往后遍历最优制作日，计算可制作数量
    for (int i = 1; i <= n; i++) {
        if (is_best[i]) {  // 若当前天是最优制作日
            // 积累的柠檬数 = 距离上一次制作的天数（每天新增1个）
            lemons += i - last_day;
            // 更新上一次制作的天数
            last_day = i;
            // 用积累的柠檬制作尽可能多的柠檬水（整数除法）
            total_cups += lemons / cost[i];
            // 记录剩余柠檬（用于下次积累）
            lemons %= cost[i];
        }
    }
    // 输出最多可制作的柠檬水杯数
    cout << total_cups;
    return 0;
}
```