讲评课

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.12

启航舱
中小学拔尖创新人才培养中心

目录

01

A

TEACHING
COURSEWARE

TEACH

A

题目描述

八尾勇喜欢吃苹果。她现在有 m（$1 \leq m \leq 100$）个苹果，吃完一个苹果需要花费 t（$0 \leq t \leq 100$）分钟，吃完一个后立刻开始吃下一个。现在时间过去了 s（$1 \leq s \leq 10000$）分钟，请问她还有几个完整的苹果？

输入格式

输入三个**非负整数**表示 m, t, s。

输出格式

输出一个整数表示答案。

输入数据 1

```
50 10 200
```

Copy

输出数据 1

```
30
```

Copy

提示

如果你出现了 RE，不如检查一下被零除？

A

这道题是求经过 s 分钟后八尾勇剩余的完整苹果数。解题时，首先要处理 t=0 的特殊情况，此时苹果会被瞬间吃完，剩余 0 个；
当 t>0 时，若 s 能被 t 整除，说明 s 分钟内刚好吃了 s/t 个苹果，剩余苹果数为 max（m - s/t，0）；
若 s 不能被 t 整除，说明 s 分钟内吃了 s/t + 1 个苹果，剩余苹果数为 max（m - s/t - 1，0），用 max 函数保证结果不为负数。

A

```cpp
#include<bits/stdc++.h>
using namespace std;
int m, s, t;  // 定义全局变量：m(初始苹果数)、s(时间)、t(每个苹果耗时)
int main() {
    cin >> m >> t >> s;  // 输入三个整数
    // 特殊情况：如果吃一个苹果的时间为0，说明瞬间能吃完所有苹果
    if (t == 0) {
        cout << 0 << endl;
        return 0;
    }
    // 计算s分钟内最多能吃多少个苹果
    if (s % t == 0) {
        // 若时间能被每个苹果的耗时整除，说明刚好吃完s/t个苹果
        cout << max(m - s/t, 0) << endl;
    } else {
        // 若不能整除，说明吃了s/t个完整苹果后，还在吃第s/t+1个
        // 因此总共吃了s/t+1个苹果
        cout << max(m - s/t - 1, 0) << endl;
    }
    return 0;
}
```

02

B

TEACHING
COURSEWARE

TEACH

B

题目描述

为了给师生提供良好的用餐体验，洛谷小学的食堂坚持现炒、现做每一道菜肴。

洛谷小学一共有 a 名老师和 b 名学生。食堂的营养师为每位师生的用餐进行配额：

- 一名学生，一次用餐需要 R 克米饭，V 克蔬菜，M 克肉。
- 一名老师，一次用餐需要 $2R$ 克米饭，$3V$ 克蔬菜，$3M$ 克肉。

洛谷小学的食堂一天需要烹制两餐，分别为中午一餐、晚上一餐。其中，中午的一餐，学生和老师都需要，而晚上的一餐仅有老师需要。

现在请问，洛谷小学的食堂，一天要准备多少克米饭，多少克蔬菜，多少克肉呢？

输入格式

输入一行，五个正整数 a, b, R, V, M，分别表示老师的人数、学生的人数，以及一名学生一次用餐需要的米饭、蔬菜和肉的量。

输出格式

输出一行，三个正整数，分别表示洛谷小学的食堂一天要准备多少克米饭，多少克蔬菜，多少克肉。

B

输出格式

输出一行，三个正整数，分别表示洛谷小学的食堂一天要准备多少克米饭，多少克蔬菜，多少克肉。

样例 #1

样例输入 #1

```
5 10 200 100 150
```

Copy

样例输出 #1

```
6000 4000 6000
```

Copy

样例 #2

样例输入 #2

```
15 120 150 200 180
```

Copy

样例输出 #2

```
27000 42000 37800
```

Copy

B

提示

【样例解释】

对于样例 1，洛谷小学有 5 个老师和 10 个学生。每天每个学生吃 1 餐，每个老师吃 2 餐，因此：

- 一个学生一餐吃 200 克米饭。因此需要准备：$1 \times 10 \times 200 + 2 \times 5 \times 400 = 6000$ 克米饭。
- 一个学生一餐吃 100 克蔬菜。因此需要准备：$1 \times 10 \times 100 + 2 \times 5 \times 300 = 4000$ 克蔬菜。
- 一个学生一餐吃 150 克肉，因此需要准备：$1 \times 10 \times 150 + 2 \times 5 \times 450 = 6000$ 克肉。

因此，输出 $6000, 4000, 6000$。

【数据范围】

对于所有数据，保证：$1 \leq a, b, R, V, M \leq 10000$。

B

这道题需计算食堂一天所需的米饭、蔬菜和肉的总量。解题关键是明确师生用餐次数及各自的配额：学生仅中午用餐 1 次，按学生标准（R 克米饭、V 克蔬菜、M 克肉）计算；老师中午和晚上各用餐 1 次，共 2 次，每次按老师标准（2R 克米饭、3V 克蔬菜、3M 克肉）计算。分别算出米饭、蔬菜、肉的总量相加即可。

B


```cpp
#include<iostream>
using namespace std;
int main(){
    // 定义变量：a为老师人数，b为学生人数，r、v、m分别为学生一餐所需米饭、蔬菜、肉的量
    int a,b,r,v,m;
    // 定义变量分别存储一天所需米饭、蔬菜、肉的总量
    int mi,ve,me;
    // 输入五个正整数
    cin>>a>>b>>r>>v>>m;
    // 计算米饭总量：老师2餐× 2R/餐 + 学生1餐× R/餐
    mi=a*2*2*r + b*r;
    // 计算蔬菜总量：老师2餐× 3V/餐 + 学生1餐× V/餐
    ve=a*2*3*v + b*v;
    // 计算肉总量：老师2餐× 3M/餐 + 学生1餐× M/餐
    me=a*2*3*m + b*m;
    // 输出结果
    cout<<mi<<" "<<ve<<" "<<me;
    return 0;
}
```

03

C

TEACHING
COURSEWARE

TEACH

C

题目描述

FJ 正在考虑改变他给奶牛挤奶的时候分配牛奶桶的方式。他认为这最终能使得他使用数量更少的桶，然而他不清楚 具体是多少。请帮助他!

FJ 有 n 头奶牛（ $1 \leq n \leq 100$ ），方便起见编号为 $1 \cdots n$ 。 第 i 头奶牛需要在时间段 $[s_i, t_i]$ 之间占用 b_i 个 桶来挤奶，此时若有其它奶牛也在挤奶则只能使用其它的桶。为了简化工作流程，FJ保证在任一时刻，至多只有一 头奶牛开始或是结束挤奶（也就是说，所有的 s_i 和 t_i 各不相同）。

现在 FJ 想考考你，他至少需要准备多少个桶，就可以满足所有奶牛的挤奶工作不会发生冲突?

输入格式

第一行输入一个正整数 n 。

之后 n 行，每行三个数 s_i, t_i, b_i 描述一头奶牛的挤奶需求。 其中 $1 \leq s_i < t_i \leq 1000$， $1 \leq bi \leq 10$ 。

输出格式

输出一个整数，表示FJ需要的桶的数量。

C

样例输入

```
3
4 10 1
8 13 3
2 6 2
```

Copy

样例输出

```
4
```

Copy

数据范围

对于 100% 的数据：$1 \leq n \leq 100, 1 \leq s_i$ 。

样例解释

在这个例子中，FJ需要 4 个桶：他用桶 1 和桶 2 来给奶牛 3 挤奶（从时间 2 开始）。他用桶 3 给奶牛 1 挤奶（从时间 4 开始）。当奶牛 2 在时间 8 开始挤奶时，桶 1 和桶 2 可以再次利用，然而桶 3 不可以，所以他会使用桶 1 、桶 2和桶 4 。

C

```cpp
#include <bits/stdc++.h>
using namespace std;
int a[1002]; // 下标从1开始，大小设为1002避免越界
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        int s, t, b;
        cin >> s >> t >> b;

        // 区间[s, t)内每个时间点累加b，下标从1开始
        for (int time = s; time < t; time++) {
            a[time] += b;
        }
    }
    int ans = 0;
    // 从1开始遍历时间点
    for (int time = 1; time <= 1000; time++) {
        ans = max(ans, a[time]);
    }
    cout << ans << endl;
    return 0;
}
```

C

这道题通过差分法求解，核心思路是：对于每头奶牛的挤奶区间[s，t]及所需桶数bb
在差分数组的s位置加上bb（表示从此时开始需要增加这些桶），在t+1位置减去bb（表示此时结束，不再需要这些桶）；
之后通过计算前缀和，还原出每个时间点的实际桶数需求，同时跟踪其中的最大值，这个最大值就是满足所有奶牛挤奶需求所需的最少桶数。
将区间操作转化为两点操作，再用前缀和还原

C

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1007;  // 定义数组大小，确保覆盖所有可能的时间点
int maxn=-1;       // 记录最大桶数需求，初始化为-1
int b[N];          // 差分数组，用于记录桶数的变化
int n;             // 奶牛的数量
int main()
{
    cin>>n;        // 输入奶牛数量
    // 处理每头奶牛的挤奶需求
    for(int i=1;i<=n;i++)
    {
        int s,t,bb;
        cin>>s>>t>>bb;  // 输入开始时间、结束时间和所需桶数
        // 差分法核心：在区间起点增加桶数，在区间终点后减少桶数
        b[s] += bb;     // 从s时间开始，需要多bb个桶
        b[t+1] -= bb;   // 到t时间结束，不再需要这bb个桶（t+1开始减少）
    }
    // 计算前缀和，得到每个时间点的实际桶数，并记录最大值
    for(int i=1;i<=1000;i++)
    {
        b[i] += b[i-1];  // 前缀和：当前时间点的桶数 = 上一时间点桶数 + 变化量
        maxn = max(b[i], maxn);  // 更新最大桶数
    }
    cout<<maxn;  // 输出结果
    return 0;
}
```

04

D

TEACHING
COURSEWARE

TEACH

D

装箱

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

题目描述

Alice 喜欢吃零食。她想把零食装到箱子里去。

Alice 现在有 m 个箱子，每个箱子的容积为 k。Alice 有 n 份零食。每个零食会有占据的容积a_i。Alice 将采取这样的策略装入箱子中：

- 若当前的箱子中剩余的容积可以装入这份零食，将这份零食装入；
- 否则，将零食装入下一个箱子。

显然，Alice 有可能会遇到麻烦：当前的零食已经没有箱子可以装了。所以 Alice 会一开始主动地放弃掉前若干份零食，**从第 k 份零食再开始装箱，在此之前的 k-1 份零食全部丢弃**，以便使得剩余的零食全部可以装入到箱子之中去。Alice 想要最大化能够装入箱子的零食数量。

例如下面这个例子：

Alice 现在有 2 个箱子，每个箱子的容积为 4。

Alice 有 5 个零食，所占据的体积分别为{1，1，3，2，2}

此时，Alice 放弃掉第一份零食，从第二份开始装入箱子，第一个箱子装入第二份和第三份零食，第二个箱子装入第三份和第四份零食。这样就可以装入四份零食。因此，在这个例子中，Alice 所可以最多装入箱子的零食数量为 4。

你的任务是帮助 Alice 计算她可以装入箱子的最多零食数量。

D

输入格式

第一行三个正整数 n,m,k，以空格隔开，分别表示零食份数，箱子数量，每个箱子容积。接下来一行 n 个空格隔开的正整数，依次表示每份零食的容积。保证零食的容积不会大于单个箱子的体积。

输出格式

一行一个正整数，表示 Alice 最多可以向箱子中放入多少零食。

样例 #1

样例输入 #1

```
5 2 4
1 1 3 2 2
```

样例输出 #1

```
4
```

样例 1 解释

参考题面中的解释。

大样例见文件 pack.in/out

D

思路一：

我们不妨假设从第x个零食开始，能够正常地塞进去这k个箱子。那么，从x+1个零食，x+2个零食...开始都肯定是可以的（因为占用的空间只能更少不可能更多）。这里我们就发现可以进行二分。

二分的是从哪个零食开始可以满足装箱要求，检查的过程是O(n)的（根据题目模拟即可）

总复杂度O(nlogn)

D


```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long
int n, m, k, a[100015];  // 所有整数变量均为long long类型
// 检查从第 mid 份零食开始装，能否用不超过 m 个箱子装完后续零食
bool check(int mid) {
    int cnt = 0;  // 记录使用的箱子数量
    int sum = 0;  // 记录当前箱子剩余的容积
    for (int i = mid; i <= n; i++) {  // 从第 mid 份零食开始遍历
        if (sum < a[i]) {  // 当前箱子剩余容积装不下当前零食
            cnt++;  // 新用一个箱子
            sum = k;  // 新箱子初始容积为 k
        }
        sum -= a[i];  // 装当前零食，更新箱子剩余容积
    }
    return cnt <= m;  // 判断使用的箱子数是否不超过 m 个
}
```

D


```
signed main() {
    // 输入数据：零食份数 n、箱子数量 m、每个箱子容积 k
    cin >> n >> m >> k;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];  // 输入每份零食的容积
    }
    // 二分查找：找最小的起始位置 mid，使得从 mid 开始装能用 <= m 个箱子装完
    int l = 1, r = n, ans = n;
    while (l <= r) {
        int mid = (l + r) >> 1;  // 取中间位置
        if (check(mid)) {  // 如果从 mid 开始装可行
            ans = mid;  // 更新最小可行起始位置
            r = mid - 1;  // 尝试找更靠前（更小）的起始位置
        } else {
            l = mid + 1;  // 起始位置太靠前不行，尝试更靠后（更大）的
        }
    }
    // 计算并输出结果：n - ans + 1 是从 ans 开始能装的零食数量
    cout << n - ans + 1 << '\n';
    return 0;
}
```

D

思路二：

不妨改成倒着塞进去。直到不能再塞下零食。容易发现如果倒着有解，则正着同样有解。如果倒着无解，则正着同样无解。（想一想，为什么？）

直接倒着塞到满了k个箱子且不能再塞即可。
总复杂度O(n)

D


```cpp
#include<bits/stdc++.h>
using namespace std;
const int maxn = 2e5 + 10;
int n, m, k, a[maxn];
int cnt, tot;  // cnt记录用掉的箱子数，tot记录当前箱子已装零食的总容积
int main() {
    // 输入数据：零食份数n、箱子数量m、每个箱子容积k，以及每份零食的容积
    cin >> n >> m >> k;
    for(int i = 1; i <= n; ++i) cin >> a[i];
    // 从最后一份零食开始，逆向尝试装箱
    for(int i = n; i >= 1; --i) {
        // 判断当前箱子剩余容积能否装下这份零食
        if(tot + a[i] <= k) {
            tot += a[i];  // 能装下，更新当前箱子已装容积
        } else {
            cnt++;  // 装不下，新开一个箱子
            tot = a[i];  // 新箱子装入当前这份零食
        }
        // 当用到第m个箱子时，说明从i位置开始用m个箱子能装下后续零食
        if(cnt == m) {
            // n - i 就是从i开始到最后的零食数量，输出结果
            cout << n - i << '\n';
            return 0;
        }
    }
    // 如果遍历完所有零食，箱子都没用完m个，说明所有零食都能装下
    cout << n << '\n';
    return 0;
}
```