

## Задача А. Одежда, сапоги и мотоцикл

Расстояние от точки, где Терминатор выпал из будущего до бара, более близкого к началу улицы, равно  $l_1 = m \bmod d$  м, а до бара более близкого к концу улицы —  $l_2 = d - l_1$ . Ответ — минимум из  $l_1$  и  $l_2$ .

## Задача В. Разводим кроликов

### Подзадача 1

В рамках ограничений этой подзадачи уместным будет прямое моделирование процесса. Для больших значений время моделирования может быть неприемлемо большим.

### Подзадача 2

Из-за вполне конкретной зависимости потребности кроликов в еде, можно определить, сколько травы им потребуется к концу  $n$ -го дня:

$$f + (f + 1) + (f + 2) + \dots + (f + (n - 1)) = \frac{f + (f + n - 1)}{2}n = \frac{2f + n - 1}{2}n.$$

За время, прошедшее до утра  $n$ -го дня, всего вырастет  $gn$  килограмм травы. Стало быть, надо найти наименьшее целое решение неравенства

$$\begin{aligned} gn < \frac{2f + n - 1}{2}n &\Rightarrow g < \frac{2f + n - 1}{2} \Rightarrow \\ &\Rightarrow 2g < 2f + n - 1 \Rightarrow n > 2(g - f) + 1 \Rightarrow n = 2(g - f) + 2. \end{aligned}$$

Особый случай:  $g < f$ , то есть травы не хватит уже на первый день питания кроликов. Соответственно, в этом случае ответ равен 1.

## Задача С. Странный калькулятор

Как следует из обсуждения наиболее эффективного алгоритма, полностью решающего задачу, — обсуждение подзадачи 3, в случае корректности входных данных решение единственно.

### Подзадача 1

В этой подзадаче можно просто перебрать все числа заданной разрядности и прямым формированием соответствующих слагаемых и их суммированием найти то число, которое даёт нужную сумму.

### Подзадача 2

В рамках данных ограничений перебор всех чисел, конечно, уже невозможен.

Заметим, что если дано число, то относительно легко посчитать сумму слагаемых, из него получающихся, и проверить, подходит число или нет. Кроме того, имеем, что при увеличении исходного числа увеличивается и сумма, ему соответствующая. (Что, кстати, тоже обосновывает единственность нужного числа, если оно существует.)

Соответственно, можно применить алгоритм двоичного поиска для нахождения нужного числа, начав с отрезка чисел от  $10^{n-1}$  (сумма, соответствующая которому, меньше или равна  $S$ ) до  $10^n - 1$  (сумма, соответствующая которому, больше или равна  $S$ ). Здесь нужно реализовать операции длинной арифметики — зануление нужного количества младших разрядов, сложение, сравнение — или воспользоваться функциями из подходящей библиотеки (если она доступна для вашего языка).

### Подзадача 3

Наиболее полный алгоритм, имеющий линейную сложность по длине искомого числа, опирается на математические рассуждения о природе задачи.

Рассмотрим сумму в столбик всех рассматриваемых чисел:

$$\begin{array}{r}
 \overline{a_n a_{n-1} a_{n-2} \dots a_4 a_3 a_2 a_1} \\
 \overline{a_n a_{n-1} a_{n-2} \dots a_4 a_3 a_2 0} \\
 \overline{a_n a_{n-1} a_{n-2} \dots a_4 a_3 0 0} \\
 + \overline{a_n a_{n-1} a_{n-2} \dots a_4 0 0 0} \\
 \dots\dots\dots \\
 \overline{a_n a_{n-1} 0 \dots 0 0 0 0} \\
 \overline{a_n 0 0 \dots 0 0 0 0} \\
 \hline
 s_{n+1} s_n s_{n-1} s_{n-2} \dots s_4 s_3 s_2 s_1
 \end{array}$$

Здесь  $a_i$ ,  $i = 1, \dots, n$ , — цифры исходного числа,  $s_j$ ,  $j = 1, \dots, n$  — цифры суммы  $S$ , объект  $s_{n+1}$  потенциально может быть многозначным числом. Как обычно в математике, надчёркивание над переменными означает, что объекты не перемножаются, а их десятичные записи записываются подряд. Из этой записи видно, что в  $k$ -м разряде,  $k = 1, \dots, n$ , суммируются  $k$  экземпляров цифры  $a_k$ .

Обозначим через  $c_k$  перенос, который даёт сложение цифр в  $k$ -м разряде с учётом переноса из предыдущего разряда:  $10 \cdot c_k + s_k = k \cdot a_k + c_{k-1}$ . Наблюдением, ключевым для решения, является весьма нетривиальное неравенство  $c_k < k$ . Докажем его по индукции.

База:  $c_1 = 0 < 1$ , так как из разряда единиц нет никакого переноса.

Шаг: Пусть  $c_{k-1} < k - 1$ . Тогда имеем

$$\begin{aligned}
 10 \cdot c_k + s_k &= k \cdot a_k + c_{k-1} \leq k \cdot 9 + c_{k-1} = \\
 &= k \cdot (10 - 1) + c_{k-1} = 10k - k + c_{k-1} < 10k - k + k - 1 = 10k - 1.
 \end{aligned}$$

Отсюда  $c_k = (10 \cdot c_k + s_k) \operatorname{div} 10 < (10k - 1) \operatorname{div} 10 < k$ . Шаг индукции доказан, и доказано нужно неравенство.

Здесь и ниже, как принято во многих языках программирования,  $\operatorname{div}$  — операция взятия частного от деления нацело, а  $\operatorname{mod}$  — операция взятия остатка.

Рассмотрим первые начало записи суммы  $S$  — число  $\overline{s_{n+1}s_n} = 10 \cdot s_{n+1} + s_n$ . Имеем  $10 \cdot s_{n+1} + s_n = n \cdot a_n + c_{n-1}$ ; при этом  $c_{n-1} < n - 1 < n$ . Отсюда получаем

$$a_n = (10 \cdot s_{n+1} + s_n) \operatorname{div} n, \quad c_{n-1} = (10 \cdot s_{n+1} + s_n) \operatorname{mod} n.$$

Продолжая далее, получаем, что

$$\overline{c_{n-1}s_{n-1}} = 10 \cdot c_{n-1} + s_{n-1} = (n - 1) \cdot a_{n-1} + c_{n-2}.$$

То есть после суммирования  $(n - 1)$  экземпляра цифры  $a_{n-1}$  и переноса  $c_{n-2}$  из разряда  $(n - 2)$ , получаем число, имеющее в разряде единиц цифру  $s_{n-1}$ , а дальнейшие его цифры есть цифры переноса  $c_{n-1}$ . В силу наблюдения  $c_{n-2} < n - 2 < n - 1$ . Поэтому

$$a_{n-1} = (10 \cdot c_{n-1} + s_{n-1}) \operatorname{div} (n - 1), \quad c_{n-2} = (10 \cdot c_{n-1} + s_{n-1}) \operatorname{mod} (n - 1).$$

Аналогично, при рассмотрении разряда  $k$  имеем

$$\overline{c_k s_k} = 10 \cdot c_k + s_k = k \cdot a_k + c_{k-1}, \quad c_{k-1} < k - 1 < k,$$

и

$$a_k = (10 \cdot c_k + s_k) \operatorname{div} k, \quad c_{k-1} = (10 \cdot c_k + s_k) \bmod k,$$

где величина  $c_k$  определена с предыдущего шага процедуры.

Тем самым определен циклический алгоритм, который на каждом шаге устанавливает цифру очередного разряда исходного числа и перенос, который был сделан в этот разряд из разряда, соответствующего меньшей степени 10.

Например, для примера из задачи имеем  $S = 1683$ :

- 1)  $\overline{s_{n+1}s_n} = \overline{s_4s_3} = 16$ ,  $a_3 = 16 \operatorname{div} 3 = 5$ ,  $c_2 = 16 \bmod 3 = 1$ ;
- 2)  $\overline{c_2s_2} = 18$ ,  $a_2 = 18 \operatorname{div} 2 = 9$ ,  $c_1 = 18 \bmod 2 = 0$ ;
- 3)  $\overline{c_1s_1} = 03 = 3$ ,  $a_1 = 3 \operatorname{div} 1 = 3$ ,  $c_0 = 3 \bmod 1 = 0$  (впрочем,  $c_0$  можно уже не вычислять).

Таким образом, получили ответ 593, какой и должны были получить.

Единственно, в постановке задачи мы не знаем разрядность начальной части суммы  $s_{n+1}$ , поэтому нужно считать всю строку записи суммы  $S$  и определить, сколько «лишних» цифр сверх  $n$  мы имеем в начале записи числа  $S$ .

Ещё одно замечание. Поскольку мы имеем, что  $s_{n+1} = c_n < n \leq 10^5$ , то число  $s_{n+1}$  и все последующие числа входят в обычный процессорный целый тип. Поэтому все вычисления проводятся без использования библиотеки длинной арифметики.

## Задача D. Ёлки на Новый Год

### Подзадача 1

Тесты данной подзадачи могут быть решены прямым перебором всех возможных троек продавцов и выбором подходящей тройки. Сложность такого решения, очевидно, равна  $O(n^3)$ .

### Подзадача 2

Перебор из первой подзадачи можно улучшить, если перебирать номер  $j$  продавца средней ёлки, а номера  $i$  и  $k$  двух других продавцов искать слева и справа от  $j$ . Сложность такого перебора улучшается до  $O(n^2)$ .

### Подзадача 3

Улучшение алгоритма до наиболее оптимального с линейной сложностью опирается на следующее наблюдение. В рамках второй подзадачи номера  $i$  и  $k$  можно искать как номера минимума среди элементов  $h_m$ ,  $1 \leq m < j$ , и максимума среди элементов  $h_m$ ,  $j < m \leq n$ , соответственно. Лобовой поиск этих минимумов и максимумов, однако, не улучшает сложности, поскольку так же требует прохода по частям массива  $h$  до и после элемента с номером  $j$ .

Однако индексы минимумов в начальных частях массива и максимумов в конечных частях можно предвычислить за линейное время:

```
min_ind[1] := 1;
for m := 2 to n do
  if h[min_ind[m-1]] < h[m]
    then min_ind[m] := min_ind[m-1];
  else min_ind[m] := m;
end for

max_ind[n] := n;
for m := n-1 downto 1 do
  if h[max_ind[m+1]] > h[m]
    then max_ind[m] := max_ind[m+1];
```

```
    else max_ind[m] := m;
```

```
end for
```

Соответственно, поиск требуемой тройки номеров продавцов с использованием массивов `min_ind` и `max_ind` можно оформить следующим образом:

```
found := false;
```

```
i := 1;
```

```
while not found and i < n-1 do
```

```
    i++;
```

```
    if h[min_ind[i]] < h[i] and h[max_ind[i]] > h[i]
```

```
        then found := true;
```

```
end while
```

```
if found
```

```
    then print min_ind[i], i, max_ind[i];
```

```
    else print 0;
```

## Задача Е. Физический процесс

### Подзадача 1

В данных ограничениях можно перебрать *все возможные* значения  $\omega_t$  для всех имеющих  $t$  и выбрать наилучший с точки зрения показателя  $\Delta$ .

### Подзадача 2

Очевидно, в данной ситуации совместимыми с измерениями являются все константные процессы со значениями из диапазона  $[\max l_t, \min u_t]$ . Можно вывести любой из них.

### Подзадача 3

Нетрудно понять, что мы можем достаточно просто проверить, является ли значение  $\Delta$  максимального изменения показателя совместимым с данным набором измерений. Действительно, если для  $t = 1$  имеем диапазон допустимых значений  $[l_1^*, u_1^*] = [l_1, u_1]$ , то для  $t = 2$  диапазон допустимых значений не больше диапазона  $[l_1^* - \Delta, u_1^* + \Delta]$ . Однако из измерения мы знаем, что этот диапазон не больше диапазона  $[l_2, u_2]$ . Значит, наш диапазон равен пересечению этих двух отрезков:

$$[l_2^*, u_2^*] = [l_1^* - \Delta, u_1^* + \Delta] \cap [l_2, u_2].$$

Аналогично для  $t = 3$

$$[l_3^*, u_3^*] = [l_2^* - \Delta, u_2^* + \Delta] \cap [l_3, u_3],$$

и так далее. Если для какого-то  $t$  пересечение пусто, то выбранное значение  $\Delta$  несовместимо с имеющимся набором измерений. Если все отрезки  $[l_t^*, u_t^*]$  непусты, то совместимо.

Также понятно, что  $\Delta \leq \max u_t$ . Поэтому в данных ограничениях можно перебрать все значения  $\Delta$  от 0 до  $\max u_t$  и выбрать первое совместимое с имеющимся набором измерений.

Для получения какого-либо процесса, совместимого с полученным значением  $\Delta^*$ , можно провести следующие вычисления в обратном времени. Для  $t = T$  возьмём в качестве  $\omega_T^*$  какую-либо целую точку из диапазона  $[l_T^*, u_T^*]$  (поскольку по построению все они могут быть истинным значением), например,  $\omega_T^* = l_T^*$ . Далее из диапазона  $[l_{T-1}^*, u_{T-1}^*]$  возьмём какую-либо точку, отстоящую от  $\omega_T^*$  не далее, чем на  $\Delta^*$ . Например, можно было бы взять величину  $\omega_T^* - \Delta^*$ , но она может выходить за нижнюю границу допустимого диапазона  $[l_{T-1}^*, u_{T-1}^*]$ . Поэтому положим  $\omega_{T-1}^* = \max(\omega_T^* - \Delta^*, l_{T-1}^*)$ . В этом случае по построению допустимых интервалов величина  $l_{T-1}^*$  будет отстоять от  $\omega_T^*$  не далее, чем на  $\Delta^*$ . После чего повторим процесс:  $\omega_t^* = \max(\omega_{t+1}^* - \Delta^*, l_t^*)$ ,  $t = T - 1, \dots, 1$ .

## Подзадача 4

В данных ограничениях полный перебор значений  $\Delta$  невозможен. Однако имеем простое наблюдение: если какое-то  $\Delta$  является совместимым, то и все большие значения также совместимы. Поэтому оптимальное значение  $\Delta$  можно найти двоичным поиском по диапозону  $[0, \max u_i]$ .

Значения  $\omega_i^*$  строятся по алгоритму, описанному выше.

## Примечания

Подобный подход, когда вместо точечного значения какого-либо процесса мы работаем с интервальной его оценкой, связанной с возможным развитием процесса с момента предыдущего измерения и неточным измерением, полученным в текущий момент, используется в практической обработке измерений реальных процессов. Интервал возможных значений показателя называют *информационным множеством*. «Раздутый» интервал, полученный в результате пересчёта предыдущего информационного множества, называют *множеством прогноза*. А множество состояний, совместимых с очередным замером, называют *множеством неопределённости измерения*. При обработке таких интервальных оценок применяются методы теории, называемой *интервальный анализ*.