

Задача А. Собираем урожай

Вычисляем количество ходок, которые надо сделать Пете, количество ходок, которые нужно сделать брату, затем умножаем их на соответствующие времена и выбираем меньшее из результатов умножения. Некоторая тонкость состоит в вычислении количества ходок; если выражать математически, то это будут величины $\lceil n/k_1 \rceil$ и $\lceil n/k_2 \rceil$, где $\lceil \cdot \rceil$ — операция округления вверх. В большинстве языков эта операция реализована, однако лучше не прибегать к вещественным вычислениям и написать код примерно такой:

```
res := n div k1;  
if n mod k1 != 0  
  then res := res+1;
```

Задача В. Мостим дорожки

Подзадача 1

Ограничения в данной подзадаче таковы, что её можно решить прямым моделированием «жадного» выкладывания плиток. В цикле по дорожкам для очередной дорожки кладем максимально возможное количество

$$k = \min \left\{ m', \left\lfloor \frac{l}{3} \right\rfloor \right\}$$

плиток 1×3 ; здесь m' — оставшееся количество таких плиток. После чего мостим остаток дорожки $l - k$ плитками 1×1 . Вычитаем из m' количество уложенных плиток 1×3 , из n' — количество уложенных плиток 1×1 . Повторяем процесс, пока оставшиеся количества плиток n' и m' не позволят замостить ни одной дорожки.

Понятно, что сложность такого алгоритма линейна по суммарному количеству плиток и при больших значениях n и m ограничениями данный алгоритм не уложится во время

Подзадача 2

Для больших значений n и m можно провести анализ работы жадного алгоритма и сразу вычислить результат его работы.

Суммарная длина всех имеющихся плиток есть $L = n + 3m$. Стало быть, ими удастся вымостить не более $a_1 = \lfloor L/l \rfloor$ дорожек. Однако, если l некратно 3, то на каждую дорожку надо доложить $(l \bmod 3)$ плиток 1×1 . Следовательно, в этом случае количество замощённых дорожек не может превышать величины $a_2 = \lfloor n/(l \bmod 3) \rfloor$.

Таким образом, ответ равен $\min\{a_1, a_2\}$.

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.

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

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

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) $\overline{s_{n+1} s_n} = \overline{s_4 s_3} = 16$, $a_3 = 16 \operatorname{div} 3 = 5$, $c_2 = 16 \bmod 3 = 1$;
- 2) $\overline{c_2 s_2} = 18$, $a_2 = 18 \operatorname{div} 2 = 9$, $c_1 = 18 \bmod 2 = 0$;
- 3) $\overline{c_1 s_1} = 03 = 3$, $a_1 = 3 \operatorname{div} 1 = 3$, $c_0 = 3 \bmod 1 = 0$ (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. Коллекция настолок

Полное решение задачи, очевидно, требует алгоритма со сложностью $O(n)$ или $O(n \log n)$ по размеру входных данных n . Соответственно, эффективность реализации упирается в структуры данных, позволяющих достаточно эффективно обрабатывать имеющийся набор номеров игр и получать информацию о наличии возможного обмена.

Для хранения набора игр очевидно из описания ситуации следует необходимость использования *стека*.

Описание обменов представляет собой набор соответствий пара $(g_{i,1}, g_{i,2}) \mapsto r_i$, который достаточно эффективно хранится в структуре *словаря*. Заметим, что пара $(g_{i,1}, g_{i,2})$ обмениваемых игр по условию задачи неупорядочена, поэтому надо либо хранить два вхождения обмена для пар $(g_{i,1}, g_{i,2})$ и $(g_{i,2}, g_{i,1})$, либо, что чуть более эффективно, хранить только пары $(g_{i,1}, g_{i,2})$, у которых $g_{i,1} < g_{i,2}$, а при обращении к словарю сортировать величины $g_{i,1}$, $g_{i,2}$ по возрастанию.

Обработку продажи одной игры из пары одинаковых проще всего организовать через прямое сравнение на совпадение двух верхних элементов стека, коль скоро во втором подходе к хранению данных требуется обработка пары номеров игр, верхних в стопке.

Группы с тестами частичного размера введены для оценивания усилий участников, которые не знают нужных структур данных или не увидели необходимость их использования.

Задача Е. 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;
```