

## Задача А. Ещё один турнир по крестикам-ноликам

Несложно понять, что в рамках каждого тура мальчики в сумме играют 3 партии: партия, которую они играют между собой, учитывается каждым мальчиком, плюс партия, которую играет победитель в финале.

Поэтому, если сумма названных чисел не делится на 3, то ситуация точно невозможна.

### Подзадача 1

Если же сумма на 3 делится, то можно сделать заключение, что количество сыгранных туров равно  $(m + p)/3$ . Эта подзадача может быть решена простым перебором количества туров, в которых в финал вышел Петя.

### Подзадача 2

В этой подзадаче разумно ещё немного позаниматься математикой.

Пусть Петя выходил в финал  $x$  раз, а Миша —  $y$  раз. Тогда имеем линейную систему двух уравнений с двумя неизвестными  $x, y$ :

$$\begin{cases} 2x + y = p, \\ x + 2y = m. \end{cases}$$

Сложив соотношения, получим  $3(x + y) = m + p$ ,  $x + y = (m + p)/3$  (последнее число целое). Вычитая последнее соотношение из каждого из исходных, получим, что имеются целые решения  $x = (2p - m)/3$ ,  $y = (2m - p)/3$ .

Если каждое из полученных чисел неотрицательно, то ответ получен. Если хотя бы одно из чисел отрицательно, то ситуация невозможна.

Иными словами, ситуация возможна, когда решение указанной системы (которое обязательно существует!) даёт целые неотрицательные числа  $x$  и  $y$ , и невозможна, когда решения или нецелые, или отрицательные.

## Задача В. Получение уровня

Формализация задачи достаточно очевидна: в заданном массиве найдите минимальное количество последних элементов, чья сумма не меньше заданного числа, и выведите количество остальных элементов. Если сумма всех элементов массива меньше заданного числа, выведите -1.

Соответственно, алгоритм тривиален: считываем массив, идём с конца, суммируя элементы, пока не будет достигнуто требуемое значение суммы или не будет достигнуто начало массива. Данный алгоритм имеет линейную сложность и решает все подзадачи.

Особое внимание следует уделить крайним случаям  $Q = 0$  и  $E = 0$ , которые, впрочем, обрабатываются описанным алгоритмом без дополнительных его модификаций.

## Задача С. Новогоднее чаепитие

### Подзадача 1

При ограничениях данной подзадачи можно организовать «лобовую» проверку: для каждого числа из первого набора линейным поиском проверить его наличие во втором. Числа, обнаруженные в обоих наборах, записать в новый массив, который затем упорядочить и выдать.

### Подзадача 2

При ограничениях данной подзадачи «лобовой» алгоритм уже не уложится в ограничения по времени.

Данную задачу можно формализовать следующим образом: найти пересечение двух множеств чисел. Как следствие, алгоритм, привлекающий специальные структуры данных — множество (**set** во многих языках), — становится понятен. Заполняем два множества, пересекаем, выдаем.

Впрочем, множества могут представляться в компьютере в виде упорядоченных массивов. В таком представлении медленно работает вставка и удаление элемента, но нам этого не требуется. Более того, для алгоритма пересечения множеств требуется только линейный проход по массивам.

Таким образом, алгоритм без использования специальных структур данных может выглядеть следующим образом:

1. Считываем наборы и сортируем их по возрастанию (вручную или с использованием библиотечных функций).

2. Применяем алгоритм пересечения множеств, заданных упорядоченными массивами **ar1** и **ar2**, и выдающий результат в виде упорядоченного массива **arRes**:

```
i1 := 1; i2 := m; k := 0;
while i1 <= n and i2 <= m do
  if ar1[i1] < ar2[i2]
  then i1++;
  else if ar1[i1] > ar2[i2]
  then i2++;
  else begin
    if k == 0 or arRes[k] != ar1[i1]
    then k++; arRes[k] := ar1[i1];
    i1++; i2++;
  end;
end while;
```

3. Выдаём полученный массив **arRes**.

## Задача D. Экономическая катастрофа

### Подзадача 1

В данной подзадаче из-за небольшого размера входных данных возможно получить ответ прямым моделированием экономической ситуации в королевстве. Моделирование можно осуществлять с шагом в один день или большими шагами по одному сезону. Во втором случае при обнаружении нарушения ограничений по ресурсам в конце сезона нужно определить день, когда нарушение произошло в первый раз.

Однако при больших входных данных моделирование может занять слишком много времени.

### Подзадача 2

Обозначим длительность года  $L = s + a + w + p$ .

В случае больших входных данных можно прогнозировать развитие ситуации. Например, если  $w > s$ , то расход свиней будет превосходить приход и, в конце концов, неизбежно возникнет дефицит свиней. За один год потеря свиней будет равна  $(w - s)$ , стало быть, зимой года с номером  $\lceil R/(w - s) \rceil$  запас свиней будет исчерпан. Это произойдет на зимний день, следующий за днём с номером  $R \bmod (w - s)$ . А всего до этого момента пройдёт  $L \cdot (\lceil R/(w - s) \rceil - 1) + s + a + R \bmod (w - s)$  дней. Также в этом случае следует проанализировать, что на склады не удастся разместить весь приплод, полученный за первое лето, то есть проверить неравенство  $s \leq R$ . В случае его нарушения, то есть если  $s > R$ , то на  $(R + 1)$  день лета первого года произойдет катастрофа.

Если же  $w < s$ , то есть если в течение года приход свиней превалирует над расходом, то экономическая катастрофа разразится из-за того, что будет переполнен скотный двор. Это произойдет летом в году с номером  $\lceil R/(s-w) \rceil$  в день, следующий за днем с номером  $R \bmod (s-w)$ . До этого момента пройдет  $L \cdot (\lceil R/(s-w) \rceil - 1) + R \bmod (s-w)$ .

Наконец, если  $s = w$ , то в течение года приход свиней равен расходу, количество свиней в целом не изменяется, нужно только проверить, что не произойдет переполнения скотного двора первым же летом, то есть, что выполняется неравенство  $s \leq R$ . Если неравенство не выполняется, то катастрофы пройдет  $R$  дней.

Аналогичный анализ производится для зерна.

В итоге, количество дней до экономической катастрофы есть минимум из всех просчитанных периодов.

## Задача Е. Удачные последовательности игр

### Подзадача 1

В ограничениях этой подзадачи можно перебрать всевозможные разбиения исходной последовательности на три части — перебор по длине  $a$ , а потом по длине  $b$ , просчитывая сумму чисел на каждой из трёх получившихся частей и проверяя удачность хотя бы двух из них. В целом, сложность такой процедуры есть  $O(n^3)$ .

### Подзадача 2

В ограничениях этой подзадачи перебор, описанный выше, уже, вообще говоря, не уложится во ограничения по времени. Однако имеется его улучшение, связанное с классическим приёмом — *вычислением префиксных сумм*. То есть организуется массив `sums` такой, что `sums[i] = r1 + r2 + ... + ri`. Вычисляется он за линейное время, поскольку `sums[0] = 0`, `sums[i] = sums[i-1] + ri`. Соответственно, сумма элементов исходной последовательности с индексами от  $i$  до  $j$  включительно,  $1 \leq i \leq j \leq n$ , равна  $S_{i,j} = \text{sums}[j] - \text{sums}[i-1]$ .

Предвычислив массив префиксных сумм, можно оптимизировать перебор из предыдущей задачи, исключив долгое вычисление сумм каждой из последовательностей. Сложность такого алгоритма будет  $O(n^2)$ , которая обуславливается перебором  $a$  и  $b$ .

### Подзадача 3

В ограничениях этой подзадачи не работает и оптимизированный перебор из предыдущей. Надо как-то оптимизировать перебор частей, на которые разбивается исходная последовательность игр.

Здесь отдельно проверяются возможности разбиения на удачные/неудачные части:  $y(\text{дачная})-?( \text{какая-то})-\text{удачная}$ ,  $y-y-?$ ,  $?-y-y$ .

Самый простой вариант первый,  $y-?-y$ . Надо найти префикс (начальную часть) последовательности с положительной суммой и суффикс (конечную часть) также с положительной суммой; при этом требуется, чтобы они не пересекались и не касались друг друга. Поиск головы идёт линейным перебором величины  $a$  от 1 до  $n-2$ . Сумма ищется через массив префиксных сумм. Как только подходящее  $a$  найдено, переходим к поиску хвоста. Хвост также ищется за линейное время перебором  $c$  от 1 до  $n-a-1$ , и сумма на нём также считается через массив префиксных сумм. При успехе полученный вариант выдаётся.

Если разбиение вида  $y-?-y$  не нашлось, переходим к поиску разбиения вида  $y-y-?$ . Его характеристика выглядит следующим образом: имеется префикс с положительной суммой (первая часть) и более длинный префикс с большей положительной суммой (так как его сумма включает и сумму первого префикса). В терминах массива `sums` префиксных сумм это требование выражается как существование двух индексов  $1 \leq i < j < n$  таких, что  $0 < \text{sums}[i] < \text{sums}[j]$ . В частности,  $i$  может быть индексом минимальной положительной

префиксной суммы на отрезке индексов от 1 до  $(j - 1)$ . Соответственно, хотелось бы иметь массив `mins[i]`, содержащий на  $i$ -м месте значение минимума среди величин `sums[1]`, `sums[2]`, ..., `sums[i]`.

А такой массив также просчитывается за линейное время: `mins[0] = +∞`, если `sums[i] > 0`, то `mins[i] = min(mins[i - 1], sums[i])`, иначе `mins[i] = mins[i - 1]`. В качестве положительной бесконечности  $+\infty$  может выступать число, заведомо большее возможных значений префиксных сумм, например, в рамках задачи это может быть константа  $2 \cdot 10^9$ . Параллельно удобно иметь и индекс этого минимума: `ind[0] = 0`, если `mins[i] == mins[i - 1]`, то `ind[i] = ind[i - 1]`, иначе `ind[i] = i`. Видно, что массивы `mins` и `ind` также просчитываются за линейное время.

Соответственно, при наличии массивов `sums`, `mins` и `ind` задача состоит в поиске индекса  $j^*$  такого, что `sums[j*] > 0` и `sums[j*] > mins[j*]`. При этом меньший индекс равен  $i^* = \text{ind}[j^*]$ . Такой поиск делается за линейное время.

Если поиск разбиения вида  $y-y-?$  неудачен, то переходим к поиску разбиения вида  $?-y-y$ . Он похож на алгоритм поиска разбиения вида  $y-y-?$ , только опирается на структуры данных, связанные с суффиксами исходного массива: массив суффиксных сумм и массив суффиксных минимумов в массиве суффиксных сумм. Все эти массивы просчитываются также за линейное время.

Таким образом, с использованием таких структур данных перебор может быть оптимизирован для указанных ограничений на входные данные.