

Задача А. Убить босса

Нужно нанести $\lceil \frac{h}{a} \rceil$ ударов. Поскольку в языке программирования деление обычно округляет вниз, можно воспользоваться трюком, и вывести $(h + a - 1)/a$. Нетрудно убедиться, что это округление вверх.

$\lceil x \rceil$ — наименьшее целое число, которое не меньше чем x (округление вверх).

Задача В. Таблица умножения

Можно было вручную для каждого числа определить ответ, а потом в зависимости от данного числа, его вывести.

Но лучше было написать два вложенных цикла `for`, которые перебирают числа от 1 до 9, проверить произведение этих чисел на равенство n , и если хоть одна пара нашлась, вывести «Yes».

Задача С. Светофор

Необходимо было с помощью `if`-ов для каждого состояния светофора вывести следующее. Все состояния, а также какое за каким следует, было описано в условии.

Задача D. Все на своих местах

Будем последовательно обрабатывать все числа последовательности, и если i -е число не равно i , то есть карточка лежит не на своем месте, сравнивать её значение с текущими минимумом и максимумом.

Задача Е. Автобусы

Лунтик не сможет сесть в автобус в двух случаях:

1. Последний автобус уедет перед Лунтиком;
2. Количество автобусов, умноженное на максимальное количество людей в одном из них, меньше общего количества людей, которые сядут в автобус (включая Лунтика).

Также есть два случая, когда Лунтик сможет сесть в автобус:

1. $ny \leq x$ — каждый раз, когда автобус приезжает, он уезжает без свободных мест (за время x приходит хотя бы n новых пассажиров). В этом случае Лунтик садится на автобус с номером $k = \lceil \frac{t}{n} \rceil$. Раз Лунтик приходит на остановку в момент времени ty , а уезжает в kx , то ответ $kx - ty$.
2. $ny > x$ — каждый автобус заберет всех, кто в этот момент на остановке. Поскольку между заездом двух отдельных автобусов пришли люди, количество которых меньше n , то после проезда автобуса количество людей на остановке всегда равно 0. Значит нужно посчитать, сколько времени пройдет с момента прихода на остановку Лунтика до прихода очередного автобуса, это число равно $x - (ty \% x)$ (аккуратно обработать случай, когда ty кратно x , например, взять по модулю x еще и ответ).

Процентом обозначается взятие остатка от деления, а $\lceil x \rceil$ — наименьшее целое число, которое не меньше чем x (округление вверх).

Задача F. Дорога со светофорами

Для решения задачи достаточно смоделировать ситуацию. Уместно использовать два параметра: где мы находимся (позиция Лунтика) и сколько сейчас времени (время). Пока мы не дойдем до конца дороги, мы следуем алгоритму:

- Если в текущей позиции нет светофора, увеличьте позицию и время на 1.
- Если есть светофор зеленого цвета, мы увеличиваем положение и время на 1.
- Если есть красный светофор, мы увеличиваем только время на 1.

Остается только определить светофор в заданное время T . Если T начинается с нуля, то светофор красный, когда $T\%(R + G) < R$, и зеленый в противном случае. За процент обозначается операция взятия остатка от деления.

Задача G. Пасьянс

Невозможно включить только что перевернутую карту в два или более триплекса, потому что, когда такая тройка возникает, участвующие в ней карты немедленно удаляются.

Пусть число m записано на текущей перевернутой карте. Чтобы определить, можно ли включить эту карту в триплекс, необходимо знать количество перевернутых карт, на которых написаны числа m , $\frac{m}{2}$ и $2m$.

Чтобы получить текущее количество перевернутых карт, на которых написано заданное число i , будем использовать массив счетчиков cnt . В нем $cnt[i]$ — это текущее количество оставшихся перевернутых карт, на которых написано число i . При чтении числа m проверьте, возможен ли триплекс в зависимости от $cnt[m]$ и $cnt[\frac{m}{2}]$ или $cnt[m]$ и $cnt[2m]$, следя за правильностью значений $2m$ и $\frac{m}{2}$ относительно диапазонов допустимых значений. При выполнении условий возникновения триплекса в значения соответствующих элементов в массиве cnt вносятся поправки, соответствующие удалению карт, участвующих в триплексе.

Задача H. Четные цифры

Представим число как массив цифр (или строку, если она в вашем языке программирования изменяемая).

Если все цифры четные: уменьшим на 2 крайнюю правую цифру, если она больше 0: 204606 → 204604. Если крайняя правая цифра равна 0, она заменяется на 8 и продолжается до тех пор, пока не встретится цифра больше 0, из которой вычитается 2 и процесс останавливается: 204000 → 202888.

Если в номере есть нечетные цифры: крайнее левое из нечетных чисел уменьшается на 1, а все цифры после него заменяются на 8: 20412046 → 20408888. Особый случай — первая цифра равна 1, тогда мы удаляем ноль в начало числа: 11234 → 08888 → 8888

Задача I. Дальнобойщик

Если некоторые два целых числа A и B дают одинаковый остаток при делении на число C , то C делит $A - B$. Таким образом, для того, чтобы данное число M удовлетворяло требованию, оно должно разделить разности между каждой парой входных чисел. Наибольшее такое M является наибольшим общим делителем разностей входных чисел. Все остальные значения M являются делителями наибольшего M (кроме 1).

Быстро вычислить наибольший общий делитель чисел можно используя алгоритм Евклида https://e-maxx.ru/algo/euclid_algorithm

Задача J. Сортировка обменами

Будем идти по элементам последовательности слева направо, и если $a[i] \neq i$, ставить на позицию i элемент i . Для этого нам нужно уметь быстро находить, где в текущий момент находится некоторый элемент x (в нашем случае i).

Предподсчитаем массив $b[x]$, который будет хранить позицию элемента x в последовательности a . Теперь, чтобы поставить число i на свою позицию, нужно обменять элементы на позициях i и $b[i]$. Это изменение нужно не забыть отобразить в массиве b , потому что элемент, который ранее стоял на позиции i переместился.

Пример ключевой части решения на языке Python:

```
b = [0] * n
for i in range(n):
    b[a[i]] = i

diff = 0
for i in range(n):
```

```
if a[i] != i:
    a[i], a[b[i]] = a[b[i]], a[i]
    b[a[b[i]]] = b[i]
    diff += 1
```

Количество обменов очевидно оптимальное, потому что мы каждый раз обязаны сделать обмен, чтобы поставить элемент i на свою позицию, и выбора где его взять у нас нет — такой элемент только один.

Задача К. Мажоранта множества

У нас имеются последовательности a и b . Обозначим за m наименьшее количество элементов, которые должны быть равны друг другу, чтобы была мажоранта. Посчитаем массивы $A[]$ и $B[]$ — сколько раз встречается каждое значение в последовательностях a и b соответственно. При этом, если $a[i] = b[i]$, не будем подсчитывать такие элементы в B , поскольку обмен в i -м индексе не имеет смысла. Таким образом, после вычисления $A[j]$ хранит количество элементов в последовательности a , которые имеют значение, равное j .

Переберем все элементы, которые встречаются в последовательностях a и b , обозначим за x очередной элемент. Проверим, может ли он быть ответом. Если $A[x] \geq h$, то x мажоранта в исходном массиве и ответ 0. Если $A[x] + B[x] \geq h$, то для того, чтобы x стал мажорантой, нужно сделать $h - A[x]$ обменов. Сравним это число с имеющимся минимумом.

Задача Л. Изготовление моделей

Предположим, что результат (ищем, сколько секунд прошло между запуском роботов первого типа и запуском роботов второго типа) равен S . Тогда мы получаем, что роботы первого типа проработали S секунд в гараже, а роботы второго типа проработали $T - S$ секунд. Мы можем посчитать, сколько моделей роботы первого типа могут собрать за S секунд. Пусть эта сумма равна $X(S)$. Мы также можем подсчитать, сколько моделей роботы второго типа могут разрисовать и упаковать за $T - S$ секунд (при условии, что у нас неограниченное количество изготовленных моделей). Пусть эта сумма равна $Y(S)$. Если $X(S) > Y(S)$, то реальный ответ меньше S , потому что роботы второго типа не успевают разрисовать и упаковать все модельки, которые изготавливают роботы первого типа. Если $X(S) \leq Y(S)$, это означает, что ответ будет S или больше, потому что есть достаточно времени, чтобы разрисовать и упаковать все модельки. Правильный ответ — это наибольшее S , для которого $X(S) \leq Y(S)$. Чтобы найти значение S , используем двоичный поиск по величине S .

Задача М. Цепочка простого числа

Пусть MAX обозначает максимальное число, которое мы рассматриваем. Мы будем основывать решение на следующем наблюдении: максимальный размер цепочки (если она начинается с 2) не больше $2\sqrt{MAX}$. Каждое новое число большее $\sqrt{MAX}$ добавляет к сумме минимум $\sqrt{MAX}$, а значит их не больше $\sqrt{MAX}$. Чисел меньших $\sqrt{MAX}$ тоже очевидно не больше $\sqrt{MAX}$.

Итак, мы можем использовать следующий алгоритм. С помощью решета Эратосфена найдем все простые числа до MAX . Затем мы найдем все возможные цепочки (сумма которых не превышает MAX) и для каждого числа сохраним длину максимальной найденной цепочки и ее первое число. Это будет быстро, потому что мы знаем, что длина цепочки маленькая и не превосходит $2\sqrt{MAX}$. Наконец, когда мы получаем отрезок, для которого ищем самую длинную цепочку, мы просто смотрим все ответы, найденные для простых чисел на этом отрезке. Сложность решения примерно равна $O(\frac{MAX}{\log MAX} \cdot \sqrt{MAX})$, но работает очень быстро даже на Python, поскольку оценка на длину цепочек очень грубая, а простых чисел не очень много.

Познакомиться с решетом Эратосфена можно, например, тут <https://brestprog.by/topics/primesieve/>

Задача Н. Массив дедушки Шера

Разобьем массив на две доли: в одной доле будут находиться все нечетные числа, в другой доле все четные числа. Проведем ребро между числами из разных долей, если их сумма есть простое число. В построенном двудольном графе остается решить задачу о максимальном паросочетании, например, алгоритмом Куна https://e-maxx.ru/algo/kuhn_matching