Турнир Архимеда 2018. Разбор задач.

Представляем вашему вниманию разбор задач Турнира Архимеда по программированию 2018.

Турнир подготовлен под руководством Андрея Станкевича командой школьников и студентов из Санкт-Петербурга и Казани, призёров и победителей Всероссийской олимпиады и международных соревнований по информатике. Турнир готовили: Ильдар Гайнуллин, Михаил Иванов, Арсений Кириллов, Андрей Заварин, Дмитрий Гнатюк, Владимслав Мильшин и Михаил Анопренко. Турнир прорешали Александра Дроздова, Мария Федоркина, Илья Кассихин и Валерий Тепляков.

Координатор проведения на системе Яндекс.Контест: Лидия Перовская. Задачи подготовлены с помощью системы Polygon, автор Михаил Мирзаянов.

Ниже приведены разборы всех задач турнира. Везде мы старались привести как можно более простое и элегантное решение.

Разбор задачи «Решение задач»

Автор задачи:	Ильдар Гайнуллин
Подготовка тестов и решений:	Ильдар Гайнуллин и Михаил Анопренко
Автор разбора:	Михаил Анопренко

Несложно заметить, что через c дней у Фифы будет $a + c \cdot x$ решенных задач, а у Фуфы — $b + c \cdot y$ задач. Таким образом, необходимо было лишь сравнить эти две величины и в зависимости от результата сравнения вывести необходимый ответ. Приведем пример реализации решения на языке Python.

```
a = int(input())
x = int(input())
b = int(input())
y = int(input())
c = int(input())

feefa = a + c * x
foofa = b + c * y

if feefa > foofa:
    print('Feefa')
elif feefa < foofa:
    print('Foofa')
else:
    print('Draw')
```

Разбор задачи «Переводчик»

Автор задачи:	Ильдар Гайнуллин
Подготовка тестов и решений:	Ильдар Гайнуллин и Михаил Анопренко
Автор разбора:	Михаил Анопренко

В этой задаче требовалось аккуратно реализовать то, что написано в условии: заменить в строке символы по определенному правилу: букву «A» на цифру «0» и наоборот, а также букву «B» на цифру «1» и наоборот. Приведем пример реализации этого решения на языке C++.

```
#include <iostream>

using namespace std;

int main() {
    int n;
    cin >> n;
    string s;
    cin >> s;

    for (int i = 0; i < n; i++) {
        if (s[i] == 'A') {
            s[i] = '0';
        } else if (s[i] == 'B') {
            s[i] = '1';
        } else if (s[i] == '0') {
            s[i] = 'A';
        } else {
            s[i] = 'B';
        }
    }

    cout << s << endl;

    return 0;
}
```

Разбор задачи «Доставка новости»

Автор задачи: Михаил Иванов
Подготовка тестов и решений: Михаил Иванов
Автор разбора: Михаил Иванов

В этой задаче необходимо для данного клетчатого прямоугольника и двух отмеченных клеток в нём определить, может ли конь добраться от одной клетки до другой.

Решение 1. Рассмотрим несколько случаев.

1. Одна из сторон равна 1. Тогда ни с какой клетки поля невозможно сделать ход. Значит, ответ на задачу «`Yes`» ровно в том случае, если стартовая и финишная клетка совпадают.

2. Одна из сторон равна 2. Будем считать, что вертикальная сторона равна двум; другими словами, Берляндия представляется клетчатым прямоугольником из двух строчек длины n.

   Какие ходы конём допустимы в таком прямоугольнике? Ясно, что „вертикальные ходы“ (при которых номер строки меняется на 2, а номер столбца — на 1) невозможны. Кроме того, если конь находится в верхней строке, то ещё выше он оказаться не может. Аналогично, с нижней строчки нельзя шагать наверх.

   Значит, из каждой клетки возможно не более одного хода в левую сторону (и он возможен при условии, что номер столбца с конём не менее 3, иначе будет невозможно переместиться на 2 клетки влево) и не более одного хода в правую (и он возможен, если номер столбца с конём не более $n - 2$). Нетрудно видеть, что при таких операциях невозможно поменять остаток от деления величины $x + 2y$ на четыре: при каждом ходе и x, и $2y$ меняются на 2 в какую-то сторону, а если к числу два раза добавить ± 2, то его остаток от деления на 4 не изменится. Четыре возможных остатка от деления этого числа на 4 дают нам разбиение полоски $2 \times n$ на 4 группы клеток: в каждой группе от любой клетки можно добраться до любой, но переместиться из одной группы в другую невозможно.

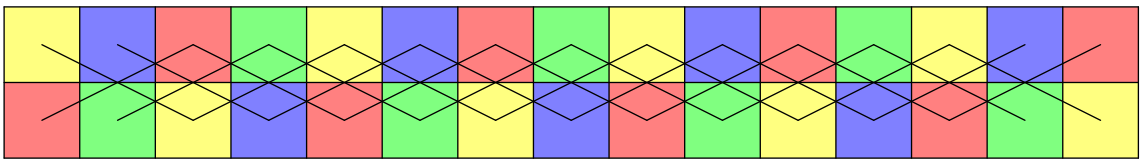

Рис. 1. Классы достижимости в прямоугольнике $2 \times n$, $n \geqslant 2$

   В единственном случае будет не четыре группы клеток, а две — при $n = 1$: тогда величина $x + 2y$ не может быть чётной.

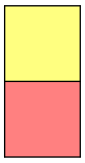

Рис. 2. Классы достижимости в прямоугольнике 2×1

   Решение в этом случае выглядит так: найти у обеих клеток величину $x + 2y$ и вывести «`Yes`», если эта величина даёт одинаковые остатки от деления на 4, и «`No`» иначе.

   Аналогично надо действовать в случае, когда двойке равна горизонтальная сторона; только тогда надо рассмотреть величину $2x + y$.

3. Обе стороны равны 3. Нетрудно убедиться, что из центральной клетки невозможно сделать ход, так что из неё достижима только она сама. Остальные клетки все достижимы друг из друга.

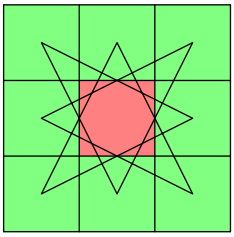

Рис. 3. Классы достижимости в квадрате 3×3

Надо выводить «`No`», если одна из клеток — $(2, 2)$, а другая — нет, в противном случае надо выводить «`Yes`».

4. Всё предыдущее неверно. Если у прямоугольника нет стороны 1, нет стороны 2 и есть сторона, не равная 3, то, очевидно, одна из его сторон не меньше 3, а другая — не меньше 4. Другими словами — наш прямоугольник содержит подпрямоугольник 3×4. Как видно из приведённого рисунка, в прямоугольнике 3×4 можно добраться от любой клетки до любой.

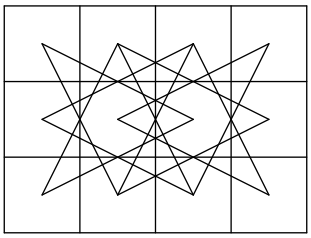

Рис. 4. Достижимость в прямоугольнике $(\geqslant 3) \times (\geqslant 4)$

Осталось понять, как это применить. Пусть нужно добраться от клетки A до клетки B; давайте же отметим много-много промежуточных клеток $C_1, \ldots, C_k$, таких что в нашем прямоугольнике есть подпрямоугольник 3×4, содержащий A и C_1; подпрямоугольник 3×4, содержащий C_1 и C_2; $\ldots$; подпрямоугольник 3×4, содержащий C_{k-1} и C_k; и, наконец, подпрямоугольник 3×4, содержащий C_k и B. Между каждой парой соседних клеток можно добраться в большом прямоугольнике, так как, как мы ранее поняли, между ними можно добраться, даже не вылезая за пределы меньшего прямоугольника 3×4. Последовательно добираясь от A до $C_1, C_2, \ldots, C_k$, в конечном итоге мы попадём в B.

Итак, в этом случае ответ всегда «`Yes`».

Асимптотика решения — $\mathcal{O}(1)$, но необходимо вручную разбирать случаи.

Решение 2.

Пусть поле имеет размеры $m \times n$. Построим граф на mn вершинах, в котором вершины соответствуют клеткам. Для каждой пары клеток, из одной из которых можно попасть в другую с помощью одного хода коня, соединим соответствующие вершины ребром. Степени всех вершин полученного графа не превосходят 8, так как из любой клетки прямоугольника за один ход можно попасть не более, чем в 8 различных клеток.

Запустим DFS на этом графе, начав из вершины, соответствующей стартовой клетке. Когда он проработает на этом графе, проверим, оказалась ли посещена вершина, соответствующая финишной клетке. Если оказалась, то до неё можно добраться ходами коня от стартовой, а иначе добраться нельзя.

К сожалению, при больших m и n в этом графе примерно $\frac{8mn}{2} = 4mn$ рёбер (в нём $(4-\varepsilon)mn$ рёбер для очень малого ε). DFS будет работать с асимптотикой $\mathcal{O}(mn)$, и, несмотря на то, что $mn \leqslant 10^8$, из-за большой константы такое решение не сможет пройти максимальный тест, уложившись в ограничение по времени.

Из этой проблемы есть выход: заметим, что если поле достаточно большое, то в нём из любой клетки можно добраться до любой. Можно, как в предыдущем решении, понять, что уже в поле

3×4 от любой до любой клетки есть путь, а можно на компьютере запустить DFS для поля 10×10 и убедиться, что любая клетка достижима из любой. Тогда, если обе стороны прямоугольника не меньше 10, то ответ точно «`Yes`»; в противном же случае в нашем прямоугольнике не более $10n$ клеток, где n — большая из сторон; а тогда в графе, соответствующем нашему полю, количество рёбер не превосходит $40n \leqslant 400000$, а DFS по такому количеству рёбер уже укладывается в ограничения. Таким образом, асимптотика $\mathcal{O}(n)$ с довольно большой константой (в зависимости от реализации и границы малости прямоугольника, роль которой у нас играло число 10), однако, поскольку n очень мало, а именно не больше 10000, решение работает быстро.

Разбор задачи «Game of stones»

Автор задачи:	Владислав Мильшин
Подготовка тестов и решений:	Ильдар Гайнуллин
Автор разбора:	Михаил Анопренко

Несложно заметить, что если в изначальной последовательности есть два соседних камня, имеющих одинаковый цвет, то ни один из этих камней никогда не сменит цвет на противоположный. Таким образом, если есть два соседних камня белого цвета, а также два соседних камня черного цвета, ответ «`No`».

Также можно убедиться, что если, например, в исходной последовательности нет двух подряд идущих камней белого цвета, то все камни можно перекрасить в черный цвет. Действительно, ведь каждый из белых камней исходно уже можно перекрасить, так как у него не может быть белых соседей. Аналогичное рассуждение верно и для черных камней. Таким образом, если есть хотя бы один цвет, для которого верно, что нет двух соседних камней этого цвета, ответ на задачу «`Yes`».

Приведем пример реализации этого решения на языке Python.

```
n = int(input())
s = input()

have00 = False
have11 = False
for i in range(n - 1):
    if s[i] == '0' and s[i + 1] == '0':
        have00 = True
    if s[i] == '1' and s[i + 1] == '1':
        have11 = True

if have00 and have11:
    print('No')
else:
    print('Yes')
```

Разбор задачи «Египетская сила!»

Автор задачи:	Дмитрий Гнатюк
Подготовка тестов и решений:	Дмитрий Гнатюк
Автор разбора:	Михаил Анопренко

В этой задаче нужно было определить, сколько различных значений может принимать сумма n слагаемых, каждое из которых принимает одно из трех значений: a, b или c. Заметим, что порядок слагаемых не имеет значения, а важно лишь количество слагаемых, равных a, количество слагаемых, равных b, и количество слагаемых, равных c. Переберем, сколько будет слагаемых a и слагаемых b. Пусть будет i слагаемых, равных a, и j слагаемых, равных b. Тогда слагаемых, равных c, будет $n - i - j$. Значит, общая сумма будет равняться $a \cdot i + b \cdot j + c \cdot (n - i - j)$. Таким образом, перебрав значения i и j, мы можем найти все возможные значения общей суммы. Чтобы определить количество различных среди этих значений, можно воспользоваться сортировкой или структурой данных set.

Асимптотика времени работы данного решения составляет $\mathcal{O}(n^2 \log n)$, так как всего существует $\mathcal{O}(n^2)$ возможных пар значений i и j, а также множитель $\log n$ мы получаем из-за сортировки или структуры данных set.

Приведем пример реализации этого решения на языке Python.

```
n = int(input())
a, b, c = map(int, input().split())
heights = set()
for i in range(n + 1):
    for j in range(n + 1):
        if i + j <= n:
            heights.add(a * i + b * j + c * (n - i - j))
print(len(heights))
```

Разбор задачи «Наблюдение на выборах»

Автор задачи:	Андрей Заварин
Подготовка тестов и решений:	Андрей Заварин
Автор разбора:	Михаил Анопренко

Для решения этой задачи воспользуемся методом динамического программирования. Пусть $dp[i][j]$ — это минимальное количество вбросов, которое произойдет на участках в первых i районах, если расставить на эти участки суммарно не более j наблюдатаелей. Изначально можно присвоить значения $dp[0][j] = 0$ для всех j, а $dp[i][j] = +\infty$ для всех $i \geqslant 1$ и j.

Пусть мы уже расставили на участки в первые i районов не более j наблюдателей. Переберем, сколько наблюдателей мы поставим на участки в районе с номером $i+1$: пусть мы хотим поставить k наблюдателей. Тогда очевидно, что ставить наблюдателей надо на участки с максимальными значениями a в этом районе. Таким образом, необходимо заранее отсортировать участки в районах по убыванию значения a, и ставить наблюдателей нужно на первые k участков в этом районе. Это позволяет нам посчитать суммарное количество вбросов в этом районе. Также не забудем, что если $k \geqslant b_{i+1}$, то на участках в этом районе не будет ни одного вброса. Таким образом, мы делаем переход из состояния $dp[i][j]$ в состояние $dp[i+1][j+k]$. Ответом на задачу будет являться значение $dp[m][C]$.

Оценим время работы программы: всего есть mC состояний и $C \cdot \sum size_i$ переходов, где $size_i$ — это количество участков в i-м районе. Так как $\sum size_i = n$, асимтотика времени работы программы составляет $\mathcal{O}(nC)$.

Разбор задачи «Сейф»

Автор задачи:	Дмитрий Гнатюк
Подготовка тестов и решений:	Владислав Мильшин
Автор разбора:	Михаил Анопренко

Пусть cnt_i — это количество раз, которое цифра i встречается в последовательности. Переберем, какая именно цифра будет встречаться в итоговой последовательности ровно k раз: пусть это цифра d. Заметим, что если $cnt_d \geqslant k$, то у нас может быть лишь излишек цифр d. Тогда нам нужно совершить $cnt_d - k$ действий для того, чтобы убрать лишние вхождения цифры d в последовательность. Если же $cnt_d < k$, то нам нужно превращать другие цифры в цифру d. Несложно заметить, что превращать цифры нужно в порядке возрастания расстояния до цифры d. Будем перебирать цифры c в порядке возрастания величины $|c-d|$, и превращать необходимое нам количество цифр c в цифры d: нужно превращать либо все имеющиеся вхождения цифры c, если их недостаточно для того, чтобы покрыть недостаток цифр d, либо ровно столько цифр c, чтобы покрыть недостаток.

Асимптотика времени работы данного решения составляет $\mathcal{O}(n + m^2)$, где m — количество различных цифр. В условиях данной задачи m — константа, равная 10.

Разбор задачи «Двоичная последовательность»

Автор задачи: Ильдар Гайнуллин
Подготовка тестов и решений: Ильдар Гайнуллин
Автор разбора: Михаил Анопренко

Решение 1. Двоичный поиск

Заметим, что если t является подпоследовательностью s_m, то она является подпоследовательностью s_n для всех $n \geqslant m$. Значит, для решения задачи можно использовать двоичный поиск по ответу. Пусть мы зафиксировали длину последовательности s_x. Тогда нам нужно проверить, что t является подпоследовательностью s_x. Для этого можно воспользоваться жадным алгоритмом: пройдем по всем символам s_x, и если очередной символ в s_x равен первому еще не встречавшемуся символу в t, то этот символ нужно взять и переместить первый не встречавшийся символ в строке t на одну позицию вправо. Если в конце прохода по строке s_x окажется, что все символы в t встречались, то t является подпоследовательностью s_x, а в противном случае — нет.

Таким образом, каждая проверка условия в двоичном поиске работает за $\mathcal{O}(n)$, а весь алгоритм имеет асимптотику времени работы $\mathcal{O}(n \log n)$, где n — длина строки t.

Решение 2. Линейный проход

Заметим, что мы можем не фиксировать заранее длину строки s_x, а искать подпоследовательность t в строке s_∞, то есть бесконечной последовательности из чередующихся цифр `0` и `1`. Тогда первый момент, когда окажется, что все символы в t уже встречались, и будет ответом на задачу.

Также можно упростить это решение, заметив, что при переходе к очередному символу в t наш алгоритм будет находить этот символ либо на следующий последовательности, если этот символ был отличен от предыдущего, либо через одну позицию, если этот символ был равен предыдущему. Таким образом, каждый символ, который равен предыдущему, увеличивает ответ на 2, а каждый символ, отличный от предыдущего, увеличивает ответ на 1. Также нужно не забыть о том, что если первый символ в t — это `0`, то ответ увеличится на 1, а если это `1`, то ответ увеличится на 2.

Данное решение работат за $\mathcal{O}(n)$. Приведем пример его реализации на языке Python.

```
t = input()

answer = 0
for i in range(len(t) - 1):
    if t[i] == t[i + 1]:
        answer += 2
    else:
        answer += 1

if t[0] == '0':
    answer += 1
else:
    answer += 2

print(answer)
```

Разбор задачи «Домашняя работа»

Авторы задачи:	Владислав Мильшин и Дмитрий Гнатюк
Подготовка тестов и решений:	Ильдар Гайнуллин
Автор разбора:	Михаил Анопренко

Заметим, что ответ «`No`» бывает лишь в том случае, когда $s_1 = s_2 = \ldots = s_n$. В этом случае можно составить лишь одно число из данных цифр. Во всех остальных случаях существует хотя бы два различных числа, и ответ на задачу «`Yes`».

Несложно доказать, что чтобы получить максимальное возможное число, которое можно составить из данных цифр, нужно расположить эти цифры в порядке убывания. Если в максимальном числе последние две цифры различны, то поменяв их местами, мы получим искомый ответ. В противном случае, посмотрим на пару цифр, стоящих перед последней цифрой. Если они различны, то поменяв их местами, мы получим искомый ответ (так как последние две цифры равны). Если же нет, то нам нужно рассмотреть предыдущую пару чисел, и так далее. Иными словами, нам нужно найти самую последнюю пару соседних различных цифр (хотя бы одна такая пара обязательно есть, ведь не все цифры одинаковы) и поменять местами эти две цифры.

Асимптотика времени работы данного решения составляет $\mathcal{O}(n \log n)$ — время на сортировку цифр. Если применить сортировку подсчетом, работающую за линейное время, можно получить решение за $\mathcal{O}(n)$.

Разбор задачи «Уборка»

Автор задачи:	Арсений Кириллов
Подготовка тестов и решений:	Арсений Кириллов
Автор разбора:	Михаил Анопренко

В данной задаче имелся большой простор для решений, необходимо было аккуратно реализовать одно из них. Рассмотрим одно из возможных решений.

Изначально переложим все книги во вторую стопку. Будем по очереди добавлять книги в первую стопку в необходимом порядке. Пусть сейчас нужно добавить в первую стопку книгу с номером x. Посмотрим, в какой из стопок лежит эта книга. Пусть, например, она лежит во второй стопке. Тогда будем перекладывать книги из второй стопки в третью, пока на верху второй стопки не окажется книга с номером x. После этого переложим эту книгу в первую стопку. Аналогичные действия совершим в случае, когда книга с номером x находится в третьей стопке.

Заметим, что для каждого значения x мы делаем не более $s+1$ действий. Значит, всего решение делает не более чем $s \cdot (s+1)$ действий, чего хватает для того, чтобы уложиться в лимит количества действий. Асимптотика времени работы данного решения составляет $\mathcal{O}(s^2)$.

Разбор задачи «Совместное счастье»

Автор задачи: Михаил Иванов
Подготовка тестов и решений: Михаил Иванов
Автор разбора: Михаил Иванов

Можно доказать, что в этой задаче работает *жадный алгоритм*, при котором Даша каждый ход совершает так, чтобы минимизировать число после этого хода, а Гена — так, чтобы максимизировать число после своего хода.

Более подробно,

- если Даше досталось ненулевое число, то оно начинается с единицы, и она её заменяет на ноль, а если ей досталось число ноль, то она заменяет его на единицу;
- если Гене досталось число, в записи которого есть ноль, то он находит самый левый ноль и заменяет его на единицу, а если ему досталось число только из единиц, то он заменяет последнюю из них на ноль.

Реализовать поиск числа, которое окажется в конце на доске, можно так. Запишем число двоичной строкой из нулей и единиц, хранящейся в памяти, и пронумеруем её символы слева направо числами от 0 до $n-1$, где n — длина исходного числа. Время от времени будем строку изменять соответственно тому, как ходят игроки. В каждый момент наша двоичная строка, если стереть у неё лидирующие нули, является числом, написанным на доске.

Заведём две переменных d и g. Переменная d — номер такого символа, что все символы левее d-го уже не участвуют в числе (то есть равны нулю и располагаются левее всех единиц строки), а g — номер такого символа, что все символы левее g-го символа или не участвуют в числе, или равны единице. Другими словами, кусок строки слева от d-го символа имеет вид $00\ldots0$, а от g-го символа — $00\ldots011\ldots1$ (нулей и единиц в каждом случае какое-то целое неотрицательное количество). Из всех таких допустимых d и g мы будем хранить наибольшие.

Изначально будем считать $d=0$. Переменную g можно приравнять к нулю и с помощью простого цикла найти первое g, что g-й символ равен нулю (или $g=n-1$, и все символы — единицы). Далее эту операцию по поиску первого подходящего g мы будем называть *простым циклом*. Пару ходов мы симулируем следующим образом.

1. По определению d указывает на первую единицу числа, если оно ненулевое, и на единственный ноль числа, если нулевое. Даша заменяет d-й символ. После этого, если он равен нулю, она увеличивает d, пока впервые не окажется, что d-й символ равен нулю или $d=n-1$. Если g оказалось меньше d, то присвоим ему значение g и простым циклом добьёмся верного значения g.
2. По определению g указывает на первый ноль числа, если оно содержит нули, и на последнюю единицу в противном случае. Гена заменяет g-й символ и простым циклом находит новое правильное g. Легко видеть, что от операции Гены правильное d не поменяется.

Сколько раз надо проделать только что описанную процедуру? Если мы повторим её k раз, то получим правильный ответ, однако в условии k ограниченно сверху числом 10^{18}, а столько операций мы позволить себе не можем. Вместо этого мы заметим две вещи: 1) каждый ход Даши уменьшает количество цифр числа на один (или, если число уже состояло из одной цифры, не меняет это количество), а каждый ход Гены оставляет количество цифр числа прежним; 2) если число состоит из одной цифры, то оно не поменяется после ходов Даши и Гены. Поэтому, если $n-1 \leqslant k$, то достаточно повторить операцию $n-1$ раз, после чего число окажется однозначным, а, значит, таким же, как и в конце игры.

Итак, нам надо повторить описанную выше операцию $\min(n-1,k)$ раз. Всегда, когда мы что-либо делаем, мы увеличиваем шаг цикла, проходящий от 1 до $\min(n-1,k)$, или увеличиваем d или g, проходящие от 0 до $n-1$. Ещё мы считываем исходное число и выводим ответ пробегом по строке за количество операций, пропорциональное n. Итого сложность работы $\mathcal{O}(n)$.

Разбор задачи «А она ему как раз»

Автор задачи:	Михаил Иванов
Подготовка тестов и решений:	Михаил Иванов
Автор разбора:	Михаил Иванов

Рассмотрим конкретную шляпу ценой c, желаемой стоимостью d, себестоимостью m со стремительностью s. Для начала предположим, что себестоимости нет, то есть шляпа в любом случае будет продана. Как тогда понять, за сколько предложений пройдёт торг?

Если $d \geqslant c$, то торг закончится немедленно, лишь только мужик скажет желаемую стоимость, то есть торг закончится за одно действие. В противном случае и мужик, и продавец назовут каждый свою сумму. Будем теперь следить за *расстоянием* — разностью между последним предложением мужика и последним предложением продавца. Изначально расстояние составит $c - d$; каждое следующее предложение, как легко видеть, уменьшает расстояние на s. Каждый из двух участников будет называть свою цену ровно в том случае, если получившееся расстояние будет строго положительным (если оно отрицательно или равно нулю, то для продавца это означает, что мужик уже предложил сумму не меньше, чем у него, а для мужика — что продавец уже предложил сумму не больше, чем у него). Итак, у нас есть число $c - d$, сколько раз можно из него вычесть s, чтобы оно осталось положительным? То есть нам нужно максимальное k, такое что $c - d - ks > 0$; $ks \leqslant c - d - 1$; $k \leqslant \frac{c-d-1}{s}$. Ответом является $\left[\frac{c-d-1}{s}\right]$, где операция $[x]$ означает нахождение *целой части* x — наибольшего целого числа, не превосходящего x.

Итак, после первых двух предложений пройдёт ещё $\left[\frac{c-d-1}{s}\right]$ реплик; итоговая длина торга равна $t = 2 + \left[\frac{c-d-1}{s}\right]$ в случае $d < c$ и $t = 1$ иначе.

Итак, пусть мы знаем t — количество реплик в торге. Как найти цену, по которой будет продана шляпа? Для начала поймём, кто выскажет последнее предложение. Так как реплики мужика и продавца чередуются, автор реплики зависит лишь от чётности t.

- t не делится на 2. Тогда автор реплики — мужик, причём это его $\frac{t+1}{2}$-я реплика. В первой реплике мужик сказал d, так что в последней реплике он скажет $d + \frac{t-1}{2} \cdot s$.
- t делится на 2. Тогда автор реплики — продавец, причём это его $\frac{t}{2}$-я реплика. В первой реплике продавец сказал c, так что в последней реплике он скажет $c - \left(\frac{t}{2} - 1\right) \cdot s$.

Значит, финальная цена шляпы f равна $d + \frac{t-1}{2} \cdot s$ при нечётных t и $c - \left(\frac{t}{2} - 1\right) \cdot s$ при чётных t.

Вспомним теперь про себестоимость m. Легко видеть, что если $m \leqslant f$, то шляпа будет продана по цене f, а иначе в процессе торгов продавец откажется её отдавать.

Итак, если мы уже имеем список данных обо всех шляпах c_i, d_i, m_i, s_i, действовать надо так.

1. Завести переменные i_{best}, f_{best}, t_{best}, отвечающие за номер, итоговую стоимость и длину торгов за оптимальную шляпу; присвоить переменной f_{best} значение, гарантированно превосходящее цену, по которой будет продана шляпа, которое мы обозначим за $+\infty$ (в задаче в качестве такого значения подойдёт, например, $10^6 + 1$).
2. Перебрать все шляпы, для каждой найти t_i, f_i описанным выше способом и, если мы сможем купить эту шляпу ($f_i \geqslant m_i$) и если она дешевле, чем шляпа, которую мы в текущий момент считаем оптимальной ($f_i < f_{\text{best}}$), то заменить значения переменных i_{best}, f_{best}, t_{best} на i, f_i, t_i.
3. Когда обработаны все шляпы, вывести «`-1`» при условии $f_{\text{best}} = +\infty$, а иначе вывести i_{best}, f_{best}, t_{best}.

Асимптотика этого решения — $\mathcal{O}(n)$.