## Решения задач

### Задача A. Перестановки.

Как понятно, любой искомый полет должен располагаться внутри отрезка, ограниченного непереносимыми полетами (X) с обоих концов, либо же ограниченного с какого-либо конца концом расписания. Если расписание не начинается с непустой последовательности из X, то добавим к нему один X слева. Если расписание не оканчивается непустой последовательностью X, то добавим один X справа. Такие дополнения избавят нас от необходимости отдельно обрабатывать первый и последний отрезки расписания, не занятые неперемещаемыми полетами (в дальнейшем будем называть их просто «отрезками расписания»).

Теперь будем последовательно максимально освобождать каждый отрезок расписания. Сначала перенесем все «переносимые» полеты как можно ближе к правому краю расписания. Для этого сначала «прижмем» к правому краю расписания все полеты на последнем отрезке (заметим, что внутри отрезка расписания их всегда можно «прижать» как к правой границе, так и к левой границе отрезка). Затем, рассматривая предпоследний отрезок, будем, начиная с самого «правого» полета (помним, что порядок менять нельзя!), переносить из него полеты в последний отрезок до тех пор, пока это возможно.

Не исключено, что нам удастся перенести из предпоследнего отрезка все полеты, а в последнем отрезке еще останется некоторое количество свободного времени. Тогда займемся перенос полетов уже из отрезка, предшествующего предпоследнему и т. д. Процесс будет продолжаться до тех пор, пока либо не закончится свободное время в последнем отрезке, либо мы не встретим переносимый полет, который в оставшееся свободное время не помещается (легко заметить, что второе условие поглощает первое).

Если переносимые полеты еще остались, продолжим их «упаковку», но в качестве целевого будем использовать уже предпоследний отрезок – пока в нем будет достаточно свободного времени, после чего перейдем к предшествующему ему отрезку и т. д.

В итоге (по рассмотрении всех переносимых полетов) мы можем быть уверены, что самый левый отрезок расписания освобожден наилучшим образом и стать «более пустым» уже не может. Количество свободного времени, оставшегося в этом отрезке и примем за «пробный максимум».

Рассмотрим второй слева отрезок расписания. Если в нем имеются какие-либо переносимые полеты, то мы можем быть уверены, что правее они передвинуты быть уже не могут. Однако мы можем попробовать перенести их влево – на самый левый отрезок расписания, рассмотренный нами на предыдущем шаге. Начнем, разумеется, с самого «левого» полета в этом отрезке и, пока будет такая возможность, будем перемещать каждый очередной полет, уменьшая количество свободного времени в самом левом отрезке (вновь заметим, что переносимые полеты в самом левом отрезке можно считать прижатыми к левому краю расписания). Когда переносить станет нечего (не останется полетов во втором слева

отрезке) или же некуда (оставшееся свободное время в самом левом отрезке не вмещает очередной полет), определяем оставшееся свободным время во втором слева отрезке и сравниваем его с пробным максимумом. При необходимости пробный максимум следует обновить.

Описанный для второго слева отрезка расписания процесс следует повторить для каждого отрезка, включая самый последний. Разумеется, когда мы будем рассматривать очередной отрезок, то для перемещения полетов из него влево можно задействовать все отрезки, которые левее него. После каждого «сдвига влево» вычисляется оставшееся свободным время на очередном отрезке. Таким образом, по завершении рассмотрения всех отрезков нам будет известно максимально возможное время полета. Чтобы знать, в какое самое раннее время полет мог начаться, необходимо сохранять позицию начала того отрезка, для которого найдено наилучшее значение по длительности полета, и полагать, что все имеющиеся полеты в этом отрезке следует «прижать» к правому краю этого отрезка — чтобы получить наиболее раннее из возможных время начала полета.

Продемонстрируем описанные выше шаги на примере из условия

```
**XAABBBAA*A*XXX*B***XX**X*
```

Сначала расписание будет преобразовано следующим образом:

```
X**XAABBBAA*A*XXX*B***XX**X*X
```

После этого в процессе перемещения всех переносимых полетов как можно ближе к правому краю расписания (при сохранении их относительного порядка) оно будем меняться так:

```
X**XAABBBAA*A*XXX*****XX**XBX
X**XAABBBAA***XXX*****XX*AXBX
X**XAABBB*****XXX***AAXX*AXBX
X***********AAXXXBBBAAXX*AXBX
```

В самом левом отрезке расписания имеется 2 единицы свободного времени. Число 2, таким образом, становится кандидатом на максимальную длительность полета. Начаться этот полет может в 0 часов.

На следующем этапе мы пробуем переместить полеты во втором слева отрезке в первый — и это позволяет освободить второй слева отрезок полностью:

```
XAAX**********XXXBBBAAXX*AXBX
```

Теперь максимально возможная длительность полета — 10. Время начала этого полета — 3. Конечно, уже понятно, что в этом небольшом расписании нет более длинных отрезков, однако проведем рассмотрение до конца. Полеты из третьего слева отрезка можно перенести во второй, что даст нам следующее расписание:

```
XAAXBBBAA*****XXX*****XX*AXBX
```

Сравнение количества свободного времени в третьем слева отрезке с текущим максимумом будет в пользу текущего максимума. Освобождение 4-го отрезка приведет к расписанию

```
XAAXBBBAAA****XXX*****XX**XBX
```

и вновь не обновит максимум. Наконец, пятый, последний отрезок, также может быть освобожден, и это также не изменит уже найденного полета максимальной длительности:

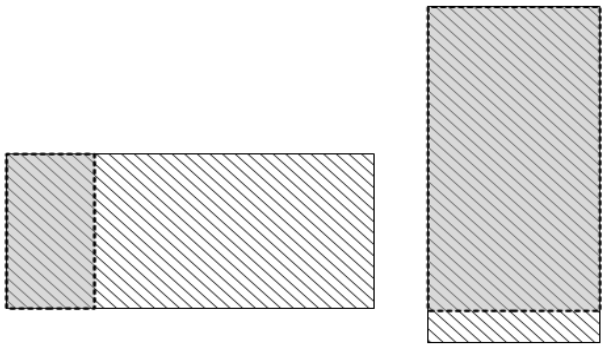
```
XAAXBBBAAAB***XXX*****XX**X*X
```

Сделаем также небольшое замечание о хранении данных. Как представляется (автору описанного решения), удобно хранить данные о перемещаемых и неперемещаемых полетах отдельно. Для первых важен взаимный порядок и принадлежность тому или иному отрезку (которая меняется по ходу действий), что позволяет вычислять оставшееся свободным время в отрезке. Вторые же фиксируют границы отрезков и используются для определения времени начала полета.

**Задача B. Сочетания.**

Важным моментом в задаче является то, что соотношение ширины и высоты любой двери задано и одинаково для всех дверей. Ковры же могут быть самых разных размеров, однако, благодаря известному соотношению между высотой и шириной двери для каждого ковра можно определить максимальные размеры двери, которую им можно завесить.

Поскольку стороны ковра должны быть параллельны сторонам двери, то, если имеется ковер со сторонами *a* и *b*, понадобится рассмотреть два положения ковра: сторона a расположена горизонтально и сторона *a* расположена вертикально (см. рис.; серая область, обведенная пунктиром, на нем — дверь максимальных размеров, которую можно завесить ковром в таком положении). В каждом из этих положений можно определить ширину двери, которую можно завесить ковром (высота определяется как удвоенная ширина). В первом положении ширина двери вычисляется как $min(a, b/2)$, во втором — как $min(b, a/2)$. Выбирая из этих двух величин максимум, получаем искомое.



Теперь нужно отсортировать двери по неубыванию ширины, а ковры — по неубыванию характеристики $max(min(a, b/2), min(b, a/2))$. «Параллельный» проход по отсортированным последовательностям позволит подсчитать пары «дверь — ковер».

Заметим, что нет необходимости искать «наиболее подходящую дверь» (или «наиболее подходящий ковер») — нас интересует максимальное количество дверей, которые могут быть завешены. Это значит, что как пара учитывается первая подходящая дверь с первым подходящим ковром, после чего происходит переход к следующей двери и следующему ковру (в упорядоченных последовательностях).

Рассмотрим пример из условия:

```
3 4
4 3 5
6 9 2 9 7 7 4 6
```

Вычислим характеристики для ковров:
$max(min(6, 9/2), min(9, 6/2)) = max(4, 3) = 4$ (учитываем лишь целую часть; у дверей целочисленные высота и ширина)
$max(min(2, 9/2), min(9, 2/2)) = max(2, 1) = 2$
$max(min(7, 7/2), min(7, 7/2)) = max(3, 3) = 3$
$max(min(4, 6/2), min(6, 4/2)) = max(3, 2) = 3$

Упорядочивая двери по ширине, а ковры по вычисленной характеристике, получаем следующие последовательности:

```
3 4 5
2 3 3 4
```

Первый ковер отбрасываем, вторым завешиваем первую дверь, третий ковер остается не использованным, четвертым завешиваем вторую дверь. Больше ковров нет, и третья дверь остается не завешенной. Таким образом, получается ответ 2.

**Задача C. Циклы**

Ограничения задачи таковы, что симуляция процесса будет неэффективна по времени: многие клетки будут закрашиваться повторно. Поэтому необходимо организовать процесс так, чтобы сэкономить на «покраске».

Одно из возможных решений выглядит следующим образом. Опишем «потенциально доступную область» как квадратную матрицу размером 4001 × 4001 (можно сделать «запас» и большим). Точка входа будет располагаться в «центре» матрицы. Такой размер гарантирует, что даже если будет сделано максимальное количество шагов с максимальным квадратом обзора в одном направлении, выхода за границы матрицы не случится.

Теперь при каждом шаге будем помечать очередной обозреваемый квадрат. Для этого в каждой строке матрицы, которую он перекрывает, в столбце, соответствующем левой границе этого квадрата будем проставлять число KK = 2*K + 1. Это значит, что, начиная с помеченной клетки, нам следовало бы закрасить KK клеток, как просмотренные. Заметим, что при движении «вверх» или «вниз» после первого шага в этом направлении достаточно будет дополнять пометками всего одну строку. При движении «вправо» или «влево» потребуется поставить такие пометки во всех перекрываемых квадратом строках матрицы.

После того, как входной файл прочитан и пометки расставлены, можно приступать к подсчету исследованной площади. Для этого нам потребуется пройти по каждой строке и «покрасить» клетки. Делается это так. Сначала двигаемся по строке, пока не обнаруживаем первую пометку. После этого записываем в счетчик число KK и продолжаем движение вправо. При этом при переходе в очередную непомеченную клетку значение счетчика (если оно положительное) уменьшается, а подсчитанная площадь увеличивается. Если при очередном переходе в непомеченную клетку счетчик становится равным нулю, то увеличение значения площади приостанавливается, а счетчик далее не уменьшается. Если же переход совершается в помеченную клетку, то текущее значение счетчика (не важно, был ли он равен нулю или же нет) заменяется значением из помеченной клетки (в нашем случае это всегда KK) и процесс продолжается.

Описанный проход по всем строкам и дает в итоге общее количество «исследованных» клеток.

Продемонстрируем описанную логику на примере из условия:

```
55 1
UUUUUURRRRRRRDDDDDDDDDDLLLLLLLLLLLLLLUUUUUURRRRRRRRRR
```

Разумеется, будет показана лишь «значащая» часть таблицы. Число KK = 3. Отметим, что при нумерации (отделена тройной линией) столбцов и строк таблицы

вполне достаточно использовать и неотрицательные числа (как это допускается в C/C++ и Java, в частности), потребуется только выполнить соответствующий сдвиг. Жирным шрифтом показаны числа, которые были «перезаписаны». Темно-серым помечен квадрат, из которого Кондрат начал свое путешествие.

| | –8 | –7 | –6 | –5 | –4 | –3 | –2 | –1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 11 | | | | | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | *2* | *1* |
| 6 | 11 | | | | | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | *2* | *1* |
| 5 | 11 | | | | | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | *2* | *1* |
| 4 | 6 | | | | | | | 3 | *2* | *1* | | | | | | 3 | *2* | *1* |
| 3 | 17 | 3 | 3 | 3 | 3 | 3 | 3 | **3** | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | *2* | *1* |
| 2 | 17 | 3 | 3 | 3 | 3 | 3 | 3 | **3** | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | *2* | *1* |
| 1 | 17 | 3 | 3 | 3 | 3 | 3 | 3 | **3** | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | *2* | *1* |
| 0 | 9 | 3 | *2* | *1* | | | | 3 | *2* | *1* | | | | | | 3 | *2* | *1* |
| –1 | 9 | 3 | *2* | *1* | | | | 3 | *2* | *1* | | | | | | 3 | *2* | *1* |
| –2 | 6 | 3 | *2* | *1* | | | | | | | | | | | | 3 | *2* | *1* |
| –3 | 17 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | *2* | *1* |
| –4 | 17 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | *2* | *1* |
| –5 | 17 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | *2* | *1* |
| –6 | | | | | | | | | | | | | | | | | | |

После того, как таблица заполнена (на рисунке нули отсутствуют), начинаем ее «закраску». Все клетки с ненулевым значением счетчика помечены серым цветом. Уменьшающееся значение счетчика записано курсивом (там, где стоят тройки, предыдущее значение счетчика, согласно алгоритму, «забывается» и он вновь становится равным 3). Бледным шрифтом без засечек написано количество закрашенных клеток в каждой из строк. Их сумму можно посчитать нарастающим итогом.

Можно обойтись без создания такой квадратной матрицы и поддерживать для каждой строки набор «закрашенных отрезков». Конечно, и в этом случае с отрезками требуется работать достаточно аккуратно: в частности, при движении по строке (в направлении R или L) следует изменять границы отрезков, а не добавлять новые; при движении по столбцу (в направлении U или D) обновлять данные лишь во вновь рассматриваемой строке и т.д. Подсчитать площадь, которую образуют закрашенные отрезки в каждой строке, можно методом «сканирующей прямой» (советуем прочитать разбор задачи E «Дорога и облака» второго тура окружного этапа Всероссийской олимпиады школьников 2010 / 2011 учебного года; он доступен по адресу
http://contest.uni-smr.ac.ru/razbor/2010/razbor_2tur_101211.pdf).

Подобный подход может сэкономить память; впрочем, это было бы существенно при ограничениях порядка $10^5$, но не $10^3$.

**Задача D. Инверсии**

Можно заметить, что описанная операция представляет собой сложение по модулю два, а эта операция коммутативна и имеет порядок 2. Обозначим через *x* количество (0 или 1) проходов через *i*-ый портал. Запишем в столбцах входные данные в виде матрицы (обозначим ее **A**); на месте символа W будет 1 а на месте B – 0. Получим условие **A****x* = **1** (mod 2), где под **1** понимается вектор-столбец, каждый элемент которого – единица. Системы уравнений по модулю 2 решаются как обычные методом Гаусса (последовательного исключения переменных). В полученном решении системы уравнений инвертируем *x*[1], поскольку этот портал был посещен сразу, и выведем *i*-ый портал в качестве ответа, если для него *x* равен 1.

**Задача E. Дополнения.**

Перевод из десятичной в какую-либо другую «обычную» (позиционную с целым положительным основанием) систему счисления выполняется делением на основание этой другой системы. Цифры в новой системе счисления получаются при этом в обратном порядке.

Пусть есть некоторое число A, записанное в десятичной системе счисления. Предположим, что мы хотим перевести его в систему счисления с основанием P. Это означает, что нужно представить число A в виде:

$$A = b_n \cdot P^n + b_{n-1} \cdot P^{n-1} + \ldots + b_2 \cdot P^2 + b_1 \cdot P^1 + b_0 \cdot P^0,$$

где $b_j$ ($j = 0, 1, \ldots, n$) — цифры в новой системе счисления ($0 \le b_j < P$).

Как можно видеть из этой формулы, остаток при целочисленном делении *A* на *P* как раз даст цифру $b_0$, а собственно целочисленное деление позволит отбросить последнюю цифру числа. Повторение целочисленного деления до тех пор, пока число *A* не обратится в ноль, и даст цифры этого числа в новой системе счисления в обратном порядке.

Покажем на примере, как переводится число 643 из десятичной системы счисления в систему счисления с основанием 7.

643 : 7 = 91 (остаток 6)
91:7 = 13 (остаток 0)
13:7 = 1 (остаток 6)
1: 13 = 0 (остаток 1)

Таким образом, $643_{10} = 1606_7$, что легко проверить прямым подсчетом:

$$1 \cdot 7^3 + 6 \cdot 7^2 + 0 \cdot 7^1 + 6 \cdot 7^0 = 1 \cdot 343 + 6 \cdot 49 + 0 + 6 = 343 + 294 + 6 = 643$$

Теперь посмотрим, чем отличается перевод в уравновешенную систему счисления от перевода в «обычную» (с таким же основанием). Отличие уравновешенной системы счисления состоит в том, что в ней используются цифры не от 0 до *P*–1 (включительно), а от –(*P*–1)/2 до (*P*–1)/2 (поэтому и основанием такой системы счисления может быть только нечетное число — как, впрочем, и

оговорено в условии задачи). Цифры от 0 до ($P$–1)/2 (включительно) совпадают в «обычной» и уравновешенной системах счисления. А вот цифры, превосходящие ($P$–1)/2, в уравновешенной системе должны «выглядеть» по-другому. Цифра $b$ ($b > (P - 1)/2$) может быть представлена в виде $P - b'$, где $b'$ как раз будет из диапазона от 1 до ($P$–1)/2 (0 не включается по причине строгого неравенства). Именно $-b'$ и станет записью цифры на этой позиции в уравновешенной системе счисления. Но к следующему разряду при этом потребуется прибавить единицу (туда «переносится» P).

Вновь вернемся к нашему примеру и продемонстрируем, как будет происходить замена цифр в уже переведенном в систему счисления с основанием 7 числе.

$$1\cdot 7^3 + 6\cdot 7^2 + 0\cdot 7^1 + 6\cdot 7^0 = 1\cdot 7^3 + 6\cdot 7^2 + 0\cdot 7^1 + (7-1)\cdot 7^0 =$$
$$= 1\cdot 7^3 + 6\cdot 7^2 + (0+1)\cdot 7^1 + (-1)\cdot 7^0 = 1\cdot 7^3 + (7-1)\cdot 7^2 + 1\cdot 7^1 + (-1)\cdot 7^0 =$$
$$= 2\cdot 7^3 + (-1)\cdot 7^2 + 1\cdot 7^1 + (-1)\cdot 7^0$$

Итак, окончательная запись числа будет выглядеть как 2–11–1 (обычно для обозначения отрицательной цифры над ней ставится черта, мы же просто соблюдаем предлагаемый в задаче формат вывода).

Разумеется, нет необходимости сначала получать запись числа в «обычной» системе счисления, а потом выполнять преобразование в уравновешенную систему счисления. Достаточно слегка модифицировать алгоритм перевода в «обычную» систему счисления. Если при переводе получается некоторая цифра $b$, большая ($P$–1)/2, то нужно сразу выполнить ее замену на $-b' = -(P - b)$, а к результату деления прибавить 1. Покажем это на примере:

643 : 7 = 91 (остаток 6). Заменяем 6 на –(7 – 6) = –1, к 91 добавляем 1
92 : 7 = 13 (остаток 1). Преобразований не требуется
13 : 7 = 1 (остаток 6). Заменяем 6 на –(7 – 6) = –1, к 1 добавляем 1
2 : 7 = 0 (остаток 2). Преобразований не требуется.
В результате получаем 2–11–1.

В заключение заметим, что хранить получаемые цифры числа удобно, например, в строке.

**Задача F. Разбиения**

Если *gcd*(A,B) != 1, то ответ, очевидно, INF. Можно доказать, что в противном случае ответ будет A·B–A–B (можно, например, воспользоваться китайской теоремой об остатках и заметить, что максимальное число имеет вид (B–1)·A + (A–1)·B – A·B = A·B – A + A·B – B – A·B = A·B – A – B)

**Задача G. Пересечения**

Понятно, что звезды с точки зрения наблюдателя на планете «сольются воедино», когда все три объекта окажутся на одной линии, проходящей через общий центр всех трех окружностей. При этом не важно, окажутся ли объекты «по одну сторону»

от этого центра (как показано на рисунке в условии) или же «по разные». Это значит, что угол отклонения от горизонтальной линии у объектов может отличаться на $180^0$.

Для удобства упорядочим объекты по угловым скоростям и введем обозначения $w1$, $w2$ и $w3$ ($w1 \le w2 \le w3$) для угловых скоростей и $f1$, $f2$, $f3$ для соответствующих начальных углов отклонения от горизонтальной линии. Теперь мы можем формализовать задачу следующим образом: хотим найти такое значение времени $t$, чтобы одновременно выполнялись следующие равенства ($k1$, $k2$, $k3$ — целые числа):

$f1 + w1 \cdot t = \varphi + 180 \cdot k1$ (1)

$f2 + w2 \cdot t = \varphi + 180 \cdot k2$ (2)

$f3 + w3 \cdot t = \varphi + 180 \cdot k3$ (3)

Исключим из рассмотрения угол $\varphi$, заменив систему из трех уравнений системой из двух уравнений: (2) – (1) и (3) – (1):

$df1 + dw1 \cdot t = 180 \cdot dk1$ (1')

$df2 + dw2 \cdot t = 180 \cdot dk2$ (2')

$df1 = f2 - f1$; $dw1 = w2 - w1$; $dk1 = k2 - k1$;

$df2 = f3 - f1$; $dw2 = w3 - w1$; $dk2 = k2 - k1$;

При этом значения $df1$ и $df2$ возьмем в диапазоне (0, 180] (обратите внимание, 0 не включается!); если это не так, всегда можно заменить их соответствующими остатками от деления; 0 просто заменяется на 180.

Перепишем полученную систему чуть иначе:

$dw1 \cdot t = 180 \cdot dk1 - df1$ (I)

$dw2 \cdot t = 180 \cdot dk2 - df2$ (II)

Сначала рассмотрим вырожденные случаи.

1. $dw1 = 0$ — два объекта движутся с одинаковыми угловыми скоростями. В этом случае если они не находятся на одной линии (проходящей через общий центр) изначально ($df1$ не равен 180) — можно сразу сообщить о неуспехе и вывести –1. Какие-либо другие выводы делать рано: нужно еще проанализировать, не совпадают ли угловые скорости у всех трех объектов.

2. $dw2 = 0$ — то же самое, что и в (1), с заменой $df1$ на $df2$).

3. $dw1 = dw2 = 0$. Все объекты должны находиться на одной линии изначально (две предыдущие проверки должны были отсечь иные случаи) и ответ — 0.

4. $dw1 = 0$, $dw2 \ne 0$. Решаем уравнение (II) относительно $t$ при условии его положительности и минимальности среди положительных.

5. $dw1 \ne 0$, $dw2 = 0$. Решаем уравнение (I) относительно $t$ также при условии, что t должно быть положительным и при этом минимальным из возможных положительных.

6. Наконец, рассмотрим невырожденный случай ($dw1 \ne 0$, $dw2 \ne 0$). Домножаем первое уравнение на $dw2$, а второе — на $dw1$. Исключая $t$, получаем:

$180 \cdot dw2 \cdot dk1 - 180 \cdot dw1 \cdot dk2 = df1 \cdot dw2 - df2 \cdot dw1$

при условии

$180 \cdot dk1 - df1 > 0$

(можно потребовать того же самого для $dk2$ и $df2$; это условие обеспечивает

положительность времени $t$). Вводя обозначения

$A = dw2 \cdot 180, \qquad B = dw1 \cdot 180, \qquad C = df1 \cdot dw2 - df2 \cdot dw1$

получаем диофантово уравнение

$A \cdot dk1 - B \cdot dk2 = C,$

которое нужно разрешить относительно $dk1$ и $dk2$.

Решение диофантова уравнения начинают с отыскания наибольшего общего делителя чисел $A$ и $B$ $gcd(A,B)$. Если найденный наибольший общий делитель также делит $C$, то уравнение имеет решение; в противном случае решения нет (и следует вывести –1).

Далее (если решение есть) следует сократить все члены уравнения на найденный $gcd(A,B)$, после чего оно может быть записано так:

$a \cdot u + b \cdot v = c,$

где $a = A/gcd(A,B)$, $b = b/gcd(A,B)$, $c = C/gcd(A,B)$, $u = dk1$, $v = -dk2$.

Поскольку $a$ и $b$ взаимно простые ($gcd(a,b) = 1$), то можно сначала найти решения уравнения $a \cdot x + b \cdot y = 1$, а затем умножить их на $c$. Решения же последнего уравнения могут быть найдены путем применения так называемого расширенного алгоритма Евклида, отыскивающего такие целые $x$ и $y$, что $a \cdot x + b \cdot y = gcd(a, b)$ (см., например, книгу «Алгоритмы: построение и анализ» Т. Кормена, Ч. Лейзерсона, Р. Ривеста и К. Штайна, книгу «Программирование: теоремы и задачи» А. Шеня; описание алгоритма несложно найти и в интернете). Мы приведем описание (псевдокод) без доказательства (которое читателям предлагается провести самостоятельно), ориентируясь на первый из указанных источников.

```
функция GCDX(a, b) возвращает (d = gcd(a,b), x, y){
    если b = 0 вернуть (a, 1, 0)
    (d', x', y') = GCDX(b, a mod b)
    (d, x, y) = (d', y', x'–|a/b|·y')
    вернуть (d, x, y)
}
```

Полученные решения $u0 = c \cdot x$ и $v0 = c \cdot y$ следует рассматривать как частные решения; общий же вид решений будет таким ($p$ — некоторый целочисленный параметр):

$u = u0 + b \cdot p$

$v = v0 - a \cdot p$

Теперь из условия положительности $dk1 = u$ нужно получить значение параметра $p$, затем собственно $dk1$. Далее из уравнения (I) вычисляется уже искомое значение времени $t$. Выводить его нужно было без какого-либо форматирования, с максимально возможной точностью. Проверяющая программа жюри сопоставляла ответы с точностью до $10^{-6}$.

В заключение заметим, что код программы несколько меньше по объему приведенного здесь описания решения.

**Задача H. Включения**

Поскольку количество входных данных очень невелико, то можно для каждой пары координат выполнить последовательно проверку для каждого круга ($x^2 + y^2 \le (k \cdot R^2)$, где $k$ — номер круга), начиная с первого. В случае успеха некоторой проверки к накопленной сумме очков добавляется соответствующее значение ($11 - k$), а дальнейшие проверки (для этой пары координат) проводить уже не нужно.

**Задача I. Объединения**

Понятно, что подсчитывать количество полос «с чистого листа» каждый раз будет очень неэффективным решением. Поэтому нужно определить, как меняется количество полос зебры при перекрашивании очередного квадрата (сами квадраты хранятся в массиве из нулей и единиц, как фактически предложено в пояснении к примерам).

Если у квадрата «разноцветные» соседи, то это означает, что перекрашиваемый квадрат (который должен был совпадать по цвету с одним из своих соседей) просто переходит из одной полосы в другую, и общее количество полос не меняется.

Если у квадрата «одноцветные» соседи, то возможны два варианта.

Первый: квадрат исходно имеет тот же цвет, что и его соседи. Это значит, что до перекрашивания они образовывали фрагмент одной полосы, и перекрашивание квадрата в другой цвет увеличивает общее количество полос на 2.

Второй: квадрат исходно имеет другой цвет, нежели его соседи. Это значит, что после перекрашивания полосы, которым принадлежат его соседи, сольются, и общее количество полос уменьшится на 2.

Разумеется, нужно аккуратно обрабатывать случаи крайних квадратов. Для этого можно, например, добавить по одному «граничному» квадрату с каждой стороны «зебры» и по завершении преобразований уменьшить соответствующим образом количество полос, если какой-либо (или оба) из этих «граничных» квадратов будет образовывать отдельную полосу.

**Задача J. Исключения.**

В условии гарантировано, что количество человек в делегации как минимум не меньше необходимого для транспортировки зеркала (этим исключается нулевой ответ).

Каждый раз, когда зеркало перемещают на K единиц, из делегации «выбывает» один человек. Чтобы зеркало было невозможно нести, в делегации должно остаться ровно M – 1 еще не напуганных. Таким образом, Рагнару нужно напугать N – (M – 1) человека, и именно столько раз зеркало будет перемещено на K единиц. Ответ вычисляется по формуле (N – M + 1) * K.