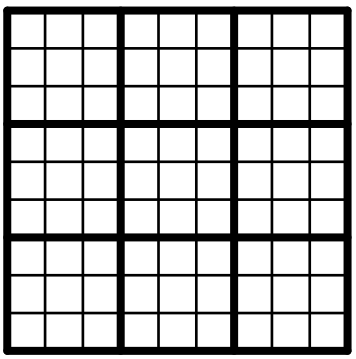
1. Проверка судоку

1 секунда 30 очков

Судоку — это головоломка, в которой игрок должен заполнить таблицу 9×9 числами $1 \ldots 9$ таким образом, чтобы в каждой строке, в каждом столбце и в каждом выделенном жирной линией на нижеприведенном рисунке квадрате 3×3 каждое из чисел присутствовало ровно один раз.



Написать программу, которая проверяет, является ли данная таблица 9×9 корректным решением головоломки судоку.

Входные данные. В текстовом файле `sk.sis` дано ровно 9 строк, в каждой строке ровно 9 целых чисел $1 \ldots 9$.

Выходные данные. На первой строке текстового файла `sk.val` вывести слово `KORRAS`, если данное во входном файле решение судоку корректно, или слово `VIGA`, если это не так. В случае ошибки, на второй и третьей строке необходимо указать координаты двух ячеек таблицы, которые содержат одинаковые числа и тем самым нарушают правила головоломки. На каждой строке вывести два отделенных пробелом числа: сначала номер строки, затем номер столбца (в обоих случаях число $1 \ldots 9$). Если противоречащих правилам пар ячеек несколько, можно вывести любую из них.

Пример.

`sk.sis`
```
1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6
2 3 4 5 6 7 8 9 1
5 6 7 8 9 1 2 3 4
8 9 1 2 3 4 5 6 7
3 4 5 6 7 8 9 1 2
6 7 8 9 1 2 3 4 5
9 1 2 3 4 5 6 7 8
```

`sk.val`
```
KORRAS
```

Пример.

`sk.sis`
```
1 2 3 4 5 6 7 8 1
4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6
2 3 4 5 6 7 8 9 1
5 6 7 8 9 1 2 3 4
8 9 1 2 3 4 5 6 7
3 4 5 6 7 8 9 1 2
6 7 8 9 1 2 3 4 5
9 1 2 3 4 5 6 7 8
```

`sk.val`
```
VIGA
1 1
1 9
```

Оценивание. В данном задании за тесты с ответом `KORRAS` очки получат лишь те программы, которые корректно решат хотя бы один тест с ответом `VIGA`.

2. XBox

1 секунда — 30 очков

В графических картах картинки обычно хранятся в виде прямоугольной мозаики (матрицы) маленьких одноцветных элементов: пикселей. Цвет каждого пикселя хранится в определенной ячейке памяти. Каждый пиксель имеет свои координаты — номер строки и столбца в матрице. У каждой ячейки памяти есть свой целочисленный адрес. Для того, чтобы передавать картинку из памяти на экран и обратно необходимо знать соответствие между адресами памяти и координатами пикселей.

У игровой консоли XBox данное соответствие определено следующим образом. Во-первых, координаты пикселя X и Y представляются в виде двоичных чисел (пусть соответствующие двоичные представления будут $X = x_n \dots x_1 x_0$ и $Y = y_n \dots y_1 y_0$). Затем две полученные битовые последовательности совмещаются поочередно, так что результат имеет вид $x_n y_n \dots x_1 y_1 x_0 y_0$. Полученная битовая последовательность интерпретируется как двоичное число Z, указывающее номер ячейки памяти, использующейся для хранения цвета искомого пикселя.

Написать программу, которая сможет определять из адреса ячейки пикселя адреса ячеек соседних пикселей.

Входные данные. На первой строке текстового файла `xb.sis` дано число Z ($0 \leqslant Z < 2^{32}$): адрес ячейки памяти пикселя $(X; Y)$. На второй два разделенных пробелом целых числа D_X и D_Y ($-1 \leqslant D_X \leqslant 1$, $-1 \leqslant D_Y \leqslant 1$), которые указывают, что необходимо найти адрес ячейки памяти для пикселя $(X + D_X; Y + D_Y)$. Можно предполагать, что $0 \leqslant X + D_X < 2^{16}$ и $0 \leqslant Y + D_Y < 2^{16}$.

Выходные данные. На единственной строке текстового файла `xb.val` вывести целое число: адрес ячейки памяти искомого пикселя.

Пример.

xb.sis	xb.val
11 -1 0	9

Адрес исходного пикселя: $11_{10} = 1011_2$. Отсюда получаем его координаты: $X = 11_2 = 3_{10}$ и $Y = 01_2 = 1_{10}$. Координаты искомого соседнего пикселя: $X - 1 = 2_{10} = 10_2$ и $Y = 1_{10} = 01_2$, а соответствующий адрес памяти: $1001_2 = 9_{10}$ (индексы около чисел здесь используются для обозначения двоичной и десятичной систем счисления).

Примечание. В позиционной системе счисления по основанию b (т.е. b-ичной системе) для обозначения чисел используются цифры $0 \dots b-1$. Последовательность цифр $a_n a_{n-1} \dots a_1 a_0$ означает число

$$a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \dots + a_1 \cdot b + a_0.$$

Поэтому цифрами двоичной системы являются 0 и 1, а значение, к примеру, двоичного числа 10101:

$$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 16 + 4 + 1 = 21.$$

3. Электронная фоторамка

1 секунда 40 очков

Электронная фоторамка — это устройство для показа загруженных в него фотографий в виде слайдшоу. Загруженные фотографии показываются по порядку. К сожалению, программное обеспечение рамки было недоделано, и работает только если загруженные файлы корректно пронумерованы по порядку начиная от 1.

Анна загрузила в свою электронную фоторамку N фотографий. Фотографии пронумерованы номерами от 1 до N, но, к сожалению, в неправильном порядке. Так как переименование загруженных в электронную фоторамку файлов достаточно неудобная процедура, то Анна хотела бы привести фотографии в правильный порядок с помощью наименьшего числа операций переименования.

Необходимо написать программу, которая находит минимальное число переименований файлов, если за раз можно переименовывать лишь один файл (поэтому, например, для того чтобы поменять имена файлов 1 и 2 друг с другом потребуются 3 операции: $1 \to 3$, $2 \to 1$, $3 \to 2$).

Входные данные. На первой строке текстового файла `dp.sis` дано целое число N ($1 \leqslant N \leqslant 1000$): количество фотографий, загруженных в рамку. На второй строке файла даны N целых чисел, разделенных пробелами. Число A_i, данное в строке на месте i показывает, что картинка, которая изначально загружена в файле i, должна после всех переименований оказаться в файле A_i. Известно, что каждое из чисел $1 \ldots N$ присутствует в строке ровно один раз.

Выходные данные. На единственной строке текстового файла `dp.val` вывести одно целое число: минимальное число операций переименования файлов. В процессе переименования файлам можно давать любые имена, но в конце должны остаться лишь номера $1 \ldots N$.

Пример.

dp.sis	dp.val
4	6
3 4 1 2	

Один из способов переименовать фотографии: $1 \to 9$, $2 \to 6$, $3 \to 1$, $4 \to 2$, $6 \to 4$, $9 \to 3$.

1. Лог-файлы

1 секунда 20 очков

Веб-сервер записывает информацию о всех приходящих запросах в *лог-файл* `access.log.0`. Когда данный файл становится слишком большим, сервер архивирует его под другим именем и продолжает писать запросы в новый пустой файл `access.log.0`. Все архивные файлы имеют имена вида `access.log.`i, причем для ограничения общего объема архива $1 \leqslant i \leqslant N$, а значения i показывают "возраст" файлов (чем новее файл, тем меньше его значение i). Иногда файлы теряются, поэтому в архиве могут отсутствовать файлы, соответствующие некоторым значениям i.

Написать программу для определения, каким образом, при добавлении нового архивного файла, необходимо переименовывать имеющиеся файлы. Процесс переименования должен удовлетворять следующим требованиям:

1. исторический порядок файлов должен сохраниться;
2. если общее число файлов вместе с новым файлом превышает N, необходимо переписать (потерять) самый старый файл;
3. нельзя терять содержание ни одного другого файла;
4. количество операций переименования должно быть минимальным.

Входные данные. На первой строке текстового файла `lf.sis` даны ограничение на суммарный размер архива N ($1 \leqslant N \leqslant 10\,000$) и количество файлов в архиве M ($0 \leqslant M \leqslant N$). На второй строке файла даны M разделенных пробелами целых чисел: номера имеющихся в архиве файлов. Известно что каждое число $1 \ldots N$ присутствует в этой строке не более одного раза.

Выходные данные. На первой строке текстового файла `lf.val` вывести число K: минимальное количество операций переименования файлов. В каждой из следующих K строк вывести два разделенных пробелом числа A_i и B_i ($0 \leqslant A_i \leqslant N$, $1 \leqslant B_i \leqslant N$, $A_i \neq B_i$, $1 \leqslant i \leqslant K$), которые описывают операцию переименования `access.log.`A_i $\rightarrow$ `access.log.`B_i. Если файл `access.log.`B_i в момент переименования уже имеется в архиве, он будет переписан и его содержимое утеряно. Если оптимальных решений несколько, можно вывести любое из них.

Пример.

`lf.sis`
```
3 3
1 2 3
```

`lf.val`
```
3
2 3
1 2
0 1
```

Новый файл можно внести в архив следующим образом:
`access.log.2` $\rightarrow$ `access.log.3`
`access.log.1` $\rightarrow$ `access.log.2`
`access.log.0` $\rightarrow$ `access.log.1`

Пример.

`lf.sis`
```
4 3
3 1 4
```

`lf.val`
```
2
1 2
0 1
```

Новый файл можно внести в архив следующим образом:
`access.log.1` $\rightarrow$ `access.log.2`
`access.log.0` $\rightarrow$ `access.log.1`

2. WiMAX

1 секунда 30 очков

Для создания сетей WiMAX необходимо устанавливать множество антенн так, чтобы между антенн была прямая видимость. Написать программу, которая получает описание ландшафта и расположение антенн и определяет на основе этих данных, между какими парами антенн есть прямая видимость, а между какими — нет.

Входные данные. На первой строке текстового файла `wm.sis` дано количество точек ландшафта N ($2 \leqslant N \leqslant 1000$) и количество антенн M ($1 \leqslant M \leqslant 1000$).

На каждой из следующих N строк дано два целых числа x_i и h_i ($0 \leqslant x_1 < x_2 < \ldots < x_N \leqslant 10\,000$, $0 \leqslant h_i \leqslant 10\,000$, $1 \leqslant i \leqslant N$), где x_i указывает горизонтальную координату точки ландшафта i, а h_i — высоту данной точки над уровнем моря. Ландшафт между заданными точками линейный.

На каждой из следующих M строк также по два целых числа x'_i ja h'_i ($x_1 \leqslant x'_1 < x'_2 < \ldots < x'_M \leqslant x_N$, $0 \leqslant h'_i \leqslant 10\,000$, $1 \leqslant i \leqslant M$), где x'_i указывает горизонтальную координату антенны i, а h'_i — высоту антенны. Можно предполагать что ни одна антенна не находится под землей.

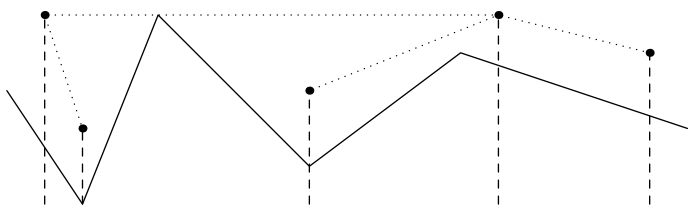
Выходные данные. В текстовый файл `wm.val` вывести ровно M строк, на каждой из них по одному целому числу. В строке i необходимо указать количество антенн, имеющих прямую видимость до антенны i. Сигнал может "касаться" поверхности ландшафта, но не может проходить насквозь. Находящаяся между двумя антеннами другая антенна не мешает связи.

Пример.

`wm.sis`
```
6 5
0 15
10 0
20 25
40 5
60 20
90 10
5 25
10 10
40 15
65 25
85 20
```

`wm.val`
```
2
1
1
3
1
```

Данный пример иллюстрирует следующая схема.



3. Чтение газет

10 секунд 50 очков

Каждый номер газеты содержит определенное число страниц. На каждой странице напечатана статья, посвященная определенному событию. Читателя интересует некоторое количество различных событий. Для того, чтобы получить информацию о них, он покупает несколько газет и начинает читать их. Каждую газету читатель читает с самого начала, подряд, страница за страницей. На чтение одной страницы читатель тратит одну минуту. Читатель заканчивает, как только доходит до страницы, после которой нет ни одной статьи о событии, о котором он не читал ранее.

Написать программу, которая укажет читателю, какие газеты и в каком порядке следует читать, чтобы получить всю интересующую его информацию за наименьшее время.

Входные данные. В первой строке текстового файла `ll.sis` дано количество купленных газет N ($1 \leqslant N \leqslant 8$), количество страниц одного номера каждой газеты M ($1 \leqslant M \leqslant 50$), и количество интересующих читателя событий K ($1 \leqslant K \leqslant N \cdot M$). На каждой из следующих N строк дано M целых чисел x_{ij} ($0 \leqslant x_{ij} \leqslant K$, $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant M$), где x_{ij} указывает номер события, о котором рассказывается на j странице газеты i. Если $x_{ij} = 0$, то это означает, что на данной странице описывается событие, которое не интересует читателя (однако, он все же должен прочитать эту страницу если в дальнейшем найдется интересующее его событие).

Выходные данные. На первой строке текстового файла `ll.val` вывести два разделенных пробелом числа: количество газет L, которое необходимо прочесть, и количество минут T, которое уйдет на чтение. На второй строке необходимо вывести L разделенных пробелами чисел: номера газет в порядке прочтения. Если возможных решений несколько, можно вывести любое из них. Можно предполагать, что каждое из событий $1 \ldots K$ описано хотя бы в одной газете.

Пример.

```
ll.sis          ll.val
4 5 3           3 5
1 0 0 0 0       1 2 3
0 2 0 0 0
0 3 0 0 0
0 2 0 0 3
```

Обратите внимание на то, что минимальным должен быть T, не L.