# Задача 1. Тупики в городе

| | |
|---|---|
| *Входной файл* | `deadends.in` |
| *Выходной файл* | `deadends.out` |
| *Ограничение по времени* | 1 сек |
| *Максимальный балл за задачу* | 100 |
| *Пример названия программы* | `p99_1.pas` |

Компания, в которой все ещё работает ваш друг, решила выпустить новую игру для мобильных телефонов, чтобы пассажирам было не так скучно стоять в пробках. Зная вас как хорошего программиста, вам поручили написать основную часть этой игры.

Игра будет состоять в том, что игрок будет управлять автобусом, перемещающимся по городу. В первой версии игры город будет представлять собой прямоугольное поле размера $N \times M$ клеток, каждая из которых либо занята зданием, либо свободна (т. е. по ней проходит дорога). Автобус игрока может перемещаться лишь по дорогам, но не по зданиям.

Кроме того, автобус считается достаточно большим, настолько, что он не может разворачиваться в пределах одной клетки. Правда, вам пока не хочется учитывать конкретные размеры автобуса, поэтому для простоты игроку будет запрещено делать два хода подряд в противоположных направлениях, а любые другие маневры будут разрешены.

Таким образом, каждым очередным ходом игрок может переместить автобус на любую соседнюю по стороне свободную клетку, кроме той, с которой автобус только что приехал. (Первым ходом можно переместить автобус в любую сторону.)

В результате понятно, что автобус игрока может застрять в тупике, откуда ему будет некуда двигаться. Более того, ясно, что есть клетки, куда заезжать нельзя, т. к., заехав туда, в итоге игрок будет вынужден доехать до тупика.

Строго говоря, пусть игрок перемещает автобус бесконечно долго (т. е. в течение бесконечного количества ходов). Тогда несложно видеть, что в некоторых свободных клетках игрок может бывать бесконечно много раз (при условии, что начальная клетка выбрана удачно), а в некоторых — не более одного (и то лишь в начальной части игры).

Сейчас вы хотите написать программу, которая разделит все свободные клетки поля на эти два типа.

## Формат входных данных

На первой строке входного файла находятся два числа $N$ и $M$ — количество строк и столбцов игрового поля соответственно ($1 \leqslant N, M \leqslant 500$). Далее следуют $N$ строк, описывающих поле. В каждой строке находятся ровно $M$ символов, каждый из которых может быть или '`#`' (решётка, ASCII-код 35), или '`.`' (точка). Решётка обозначает клетку со зданием, точка — свободную клетку.

## Формат выходных данных

В выходной файл выведите $N$ строк по $M$ символов в каждой: для клеток, в которых находятся здания, выводите '`#`', для клеток, где игрок может побывать не более одного раза — '`X`' (латинская заглавная буква X), для остальных клеток — '`.`'.

## Пример

| *Входной файл* | *Выходной файл* |
|---|---|
| 4 12<br>.#....#.##..<br>.##.#.#.##..<br>......####..<br>.##.#.#.#### | X#X...#X##..<br>X##.#.#X##..<br>XXX...####..<br>X##X#X#X#### |
| 3 2<br>##<br>##<br>## | ##<br>##<br>## |

## Система оценки

Если ваша программа будет работать при $N, M \leqslant 100$, то вы получите как минимум 50 баллов. Если ваша программа будет работать при $N, M \leqslant 250$, то вы получите как минимум 60 баллов.

Обратите также внимание, что для получения полного балла вам, скорее всего, придётся использовать 32-битный компилятор (Free Pascal, GNU C, GNU C++).

# Задача 2. Калитка в заборе

| | |
|---|---|
| *Входной файл* | `door.in` |
| *Выходной файл* | `door.out` |
| *Ограничение по времени* | 1 сек |
| *Максимальный балл за задачу* | 100 |
| *Пример названия программы* | `p99_2.pas` |

Дядя Фёдор, кот Матроскин и Шарик решили обновить забор вокруг своего сада в Простоквашино. Матроскин и Шарик, недолго думая, вкопали $N$ столбов вдоль одной из сторон участка. Это очень сильно расстроило Дядю Фёдора, так как его друзья забыли о самом главном — калитка должна находиться именно на этой стороне, и для неё необходимо было оставить проём шириной как минимум $W$. Теперь им придётся выкапывать некоторые столбы.

Чтобы работа не пропадала даром, выкопать надо как можно меньше столбов. Помогите Дяде Фёдору определить, какие именно столбы надо выкопать. После выкапывания столбов должен найтись промежуток (между двумя оставшимися столбами, или между оставшимся столбом и концом стороны участка, или между двумя концами стороны участка) ширины больше или равной $W$.

## Формат входных данных

Первая строка содержит два целых числа $N$ и $W$ — количество вкопанных столбов и минимально необходимую ширину проёма для калитки соответственно. Гарантируется, что $0 \leqslant N \leqslant 30\,000$ и что $0 \leqslant W \leqslant 60\,000$.

Будем считать, что вдоль интересующей нас стороны участка введена ось координат. Во второй строке входного файла находятся два числа $L$ и $R$ — координаты левого и правого конца этой стороны ($L < R$). Далее следуют $N$ чисел — координаты вкопанных столбов. Все координаты (включая $L$ и $R$) — различные целые числа, по модулю не превосходящие 30 000. Гарантируется, что все столбы вкопаны между левым и правым концами стороны.

## Формат выходных данных

В первой строке выходного файла должно быть минимальное число столбов, которые надо выкопать. Далее должны следовать номера этих столбов. Столбы нумеруются в том порядке, как они указаны во входном файле, начиная с 1.

Если решений несколько, то вы можете вывести любое. Если решения нет, то выведите в выходной файл одну строку, содержащую число `-1`.

## Пример

| *Входной файл* | *Выходной файл* |
|---|---|
| 3 2<br>2 6<br>3 4 5 | 1<br>2 |
| 3 2<br>1 6<br>4 3 5 | 0 |
| 3 5<br>1 7<br>5 3 4 | 3<br>2<br>1<br>3 |

# Задача 3. Множества, свободные от сумм

| | |
|---|---|
| *Входной файл* | `mss.in` |
| *Выходной файл* | `mss.out` |
| *Ограничение по времени* | 2 сек |
| *Максимальный балл за задачу* | 100 |
| *Пример названия программы* | `p99_3.pas` |

Пусть нам дано натуральное число $N$. Рассмотрим множество различных целых чисел $\{a_1, a_2, \ldots, a_k\}$, где каждое число лежит в интервале от 0 до $N-1$ включительно. Назовём такое множество *свободным от сумм*, если в этом множестве не найдётся таких трёх чисел, что сумма двух из них сравнима с третьим по модулю $N$. Строго говоря, назовём множество свободным от сумм, если для каждой тройки (не обязательно различных) индексов $x$, $y$ и $z$ ($1 \leqslant x, y, z \leqslant k$) выполняется неравенство $(a_x + a_y) \bmod N \neq a_z$, где $p \bmod q$ — остаток от деления $p$ на $q$.

Например, при $N = 6$ множествами, свободными от сумм, *не* являются, например, $\{0\}$ (т. к. $(0+0) \bmod 6 = 0$), $\{1, 2\}$ (т. к. $(1+1) \bmod 6 = 2$), $\{3, 4, 5\}$ (т. к. $(4+5) \bmod 6 = 3$), но множество $\{1, 3, 5\}$ является свободным от сумм.

По заданному $N$ определите, сколько существует множеств, свободных от сумм.

## Формат входных данных

Во входном файле находится одно целое число $N$. Гарантируется, что $1 \leqslant N \leqslant 35$.

## Формат выходных данных

В выходной файл выведите одно число — ответ на задачу.

## Пример

| *Входной файл* | *Выходной файл* |
|---|---|
| 2 | 2 |
| 6 | 14 |

*Примечание:* Все множества, свободные от сумм, для $N = 6$ — это следующие: $\{5\}$, $\{4\}$, $\{3\}$, $\{3, 5\}$, $\{3, 4\}$, $\{2\}$, $\{2, 5\}$, $\{2, 3\}$, $\{1\}$, $\{1, 5\}$, $\{1, 4\}$, $\{1, 3\}$, $\{1, 3, 5\}$, $\{\}$ (последнее множество — пустое, т.е. не содержащее ни одного элемента, с $k = 0$ — тоже считается свободным от сумм).

# Задача 4. Переливание

| | |
|---|---|
| *Входной файл* | `flow.in` |
| *Выходной файл* | `flow.out` |
| *Ограничение по времени* | 1 сек |
| *Максимальный балл за задачу* | 100 |
| *Пример названия программы* | `p99_4.pas` |

На досуге вы любите почитать сборники занимательных задач по математике. Недавно вы наткнулись в одном из таких сборников на следующую задачу:

> Есть бесконечный резервуар с водой и два пустых сосуда объёмом 5 и 12 литров. Можно: наливать воду из резервуара в любой сосуд до его заполнения, переливать воду из одного сосуда в другой до заполнения второго или опустошения первого (смотря что будет раньше) и выливать воду из сосуда на землю до полного опустошения сосуда. Как таким образом можно отмерить 3 литра?

Вы решили написать программу, которая будет решать подобные задачи для произвольных объёмов сосудов.

## Формат входных данных

Во входном файле находятся три целых числа — $V_1$, $V_2$ и $V$ — объёмы двух сосудов и объем воды, который нужно отмерить. Гарантируется, что $1 \leqslant V_1, V_2 \leqslant 32767$ и $0 \leqslant V \leqslant \max(V_1, V_2)$.

## Формат выходных данных

В первую строку выходного файла выведите одно число — количество действий в вашем решении. Далее выведите соответствующее количество строк, описывающих действия в вашем решении. Для каждого действия выведите два числа:

- если это действие — переливание из одного сосуда в другой, то первое число должно быть номером сосуда, откуда надо переливать воду, а второе — номером сосуда, куда переливать;
- если это действие — набор воды из резервуара, то первое число должно быть нулём, а второе — номером сосуда, куда наливать;
- если это действие — выливание воды «на землю», то первое число должно быть номером сосуда, а второе — нулём.

После выполнения всех операций хотя бы в одном сосуде должна находиться вода в объёме $V$.

Если существует несколько решений, то вы можете вывести любое. Ваше решение не обязано быть оптимальным, единственное ограничение — размер выходного файла не должен превосходить 3 Мб.

Если решений не существует, выведите одно число `-1`.

## Пример

| *Входной файл* | *Выходной файл* |
|---|---|
| `5 12 3` | `10`<br>`0 1`<br>`1 2`<br>`2 1`<br>`1 2`<br>`0 1`<br>`1 0`<br>`0 1`<br>`1 2`<br>`0 1`<br>`1 2` |

# Задача 5. Пересечение

| | |
|---|---|
| *Входной файл* | `sect.in` |
| *Выходной файл* | `sect.out` |
| *Ограничение по времени* | 1 сек |
| *Максимальный балл за задачу* | 100 |
| *Пример названия программы* | `p99_5.pas` |

Компания, производящая оборудование для сотовой связи, обратилась к вам с просьбой написать программу, оценивающую качество организации сети. Одним из важных параметров является то, пересекаются ли зоны покрытия передатчиков, работающих на одинаковых частотах. Для простоты будем считать, что область покрытия каждого передатчика представляет собой многоугольник на плоскости (не обязательно выпуклый). Две области покрытия будем считать пересекающимися, если у них есть хотя бы одна общая точка (возможно, лежащая на границе одной или даже обеих областей). Ваша программа должна принимать на вход набор пар многоугольников, описывающих зоны покрытия передатчиков, и выводить про каждую пару информацию о том, пересекаются ли эти зоны.

## Формат входных данных

На первой строке входного файла находится число $K$ — количество тестов во входном файле. Далее идёт описание $K$ тестов. Каждый тест задаётся описанием двух многоугольников, которые надо проверить на пересечение. Каждый многоугольник задаётся в следующем формате: сначала указывается одно число $N_i$ — число вершин этого многоугольника, — после чего идут $N_i$ строк, каждая из которых содержит два разделённых пробелом числа $x_{ij}$ и $y_{ij}$ — координаты $j$-й вершины этого многоугольника. Вершины перечислены в порядке обхода многоугольника.

Число пар многоугольников в одном тесте $1 \leqslant K \leqslant 10$, число вершин каждого многоугольника $3 \leqslant N_i \leqslant 100$, координаты вершин — целые числа, $|x_{ij}|, |y_{ij}| \leqslant 10\,000$.

## Формат выходных данных

Для каждой пары многоугольников выведите в выходной файл на отдельной строке одно слово: '`YES`', если многоугольники пересекаются, и '`NO`', если нет.

## Пример

| *Входной файл* | *Выходной файл* |
|---|---|
| 2<br>3<br>0 0<br>-1 0<br>0 -1<br>3<br>1 1<br>2 1<br>1 2<br>4<br>0 0<br>2 0<br>2 2<br>0 2<br>4<br>1 1<br>3 1<br>3 3<br>1 3 | NO<br>YES |

# Задача 6. Собрать треугольник

| | |
|---|---|
| *Входной файл* | `triangle.in` |
| *Выходной файл* | `triangle.out` |
| *Ограничение по времени* | 1 сек |
| *Максимальный балл за задачу* | 100 |
| *Пример названия программы* | `p99_6.pas` |

Вы по-прежнему работаете под руководством д. б. н., проф. О. Б. Ломова и изучаете интеллект обезьян. Ваши подопечные уже очень далеко ушли от столь элементарной задачи, как сбор квадрата. Теперь вы работаете над тем, чтобы обучить их намного более сложной задаче. Вы по-прежнему даёте обезьянам набор из $N$ палочек, но на этот раз вы хотите, чтобы они собрали из этих палочек треугольник.

Конечно, решить эту задачу в элементарном варианте — выбрать три палочки и собрать из них треугольник — ваши подопечные могут без каких-либо проблем; вы же хотите их обучить, чтобы они собирали один большой треугольник из *всех* выданных им палочек сразу. Таким образом, они должны разбить палочки на три группы так, чтобы, сложив палочки каждой группы в один большой отрезок, получить три отрезка, из которых можно собрать треугольник. Полученный треугольник должен быть невырожденным, т. е. его площадь должна быть строго больше нуля.

Как и в прошлый раз, вам понадобилась программа, которая определит, разрешима ли задача для данного набора палочек.

## Формат входных данных

На первой строке входного файла находится одно натуральное число $N$ — количество палочек в наборе ($1 \leqslant N \leqslant 16\,000$). На второй строке находятся $N$ натуральных чисел — длины палочек. Гарантируется, что суммарная длина палочек не превосходит $100\,000\,000$.

## Формат выходных данных

Если решения не существует, то в первую строку выходного файла выведите одно слово ‘`no`’ (без кавычек). В противном случае в первую строку выведите одно слово ‘`yes`’, а в следующие три строки выведите какой-нибудь способ собрать треугольник из данных палочек. Каждая из этих трёх строк должна описывать очередную сторону получающегося треугольника: в каждой строке сначала должно идти количество палочек, из которых состоит эта сторона, а потом длины этих палочек. Каждую палочку, конечно, можно использовать только один раз.

Если есть несколько способов собрать треугольник из данных палочек, выведите любой.

## Пример

| *Входной файл* | *Выходной файл* |
|---|---|
| 5<br>1 2 3 4 5 | yes<br>2 4 3<br>1 5<br>2 1 2 |
| 5<br>1 2 3 4 100 | no |