1. Фоторамка

1 секунда 10 очков

У Марии есть две фотографии, которые она хочет повесить на стену в своей комнате. Для этого ей нужно заказать фоторамку. Поскольку рейка для рамки стоит довольно дорого, Мария хочет заказать одну рамку, такую чтобы в неё помещалась каждая из двух фотографий, и на четыре края которой понадобилось бы как можно меньше рейки.

Написать программу, которая находит рамку с минимальным периметром, в которую поместится каждая фотография Марии. Рамку можно повесить на стену как вертикально (длинный край вертикально), так и горизонтально (длинный край горизонтально).

Входные данные. В первой строке текстового файла `raam.sis` даны размеры первой фотографии, во второй строке — размеры второй фотографии. Все длины — целые числа $1 \ldots 100$.

Выходные данные. В единственную строку текстового файла `raam.val` вывести два разделённых пробелом целых числа — длины сторон фоторамки. Если рамок с минимальным периметром несколько, вывести любую из них.

Пример.

`raam.sis`
```
30 30
20 40
```

`raam.val`
```
30 40
```

Размеры фотографий Марии 30×30 и 20×40. Первая (квадратная) фотография помещается в рамку 30×40 любой стороной, и от длины рамки остаётся ещё 10 cm. Вторая (прямоугольная) фотография помещается в рамку только продольно, и от длины рамки опять остаётся 10 cm. Периметр рамки 140 cm, и это самый маленький возможный периметр.

2. По росту

1 секунда 20 очков

Дан рост N учеников.

Написать программу, которая находит рост третьего по росту ученика.

Входные данные. В первой строке текстового файла `pikk.sis` дано целое число N ($3 \leqslant N \leqslant 10\,000$), а во второй строке — N целых чисел A_1, A_2, …, A_N ($0 < A_i \leqslant 1000$) — рост учеников.

Выходные данные. В единственную строку текстового файла `pikk.val` вывести одно число — рост третьего по росту ученика.

Пример.

`pikk.sis`
```
5
175 173 169 173 171
```

`pikk.val`
```
173
```

Рост самого высокого ученика 175 cm, второе и третье место делят ученики с ростом 173 cm.

3. TopSwops

1 секунда 30 очков

TopSwops — это карточная игра для одного игрока, в которую играют колодой из N карт. Карты пронумерованы числами $1 \ldots N$. Ни один номер не встречается в колоде дважды, и номер написан на обеих сторонах карты.

В начале игры карты перемешиваются и кладутся на стол одной пачкой. Затем игрок за один ход берёт сверху пачки столько карт, сколько показывает число на верхней карте, переворачивает взятую часть пачки вверх дном и кладёт обратно сверху пачки.

Понятно, что если сверху пачки окажется число 1, то дальше ничего не изменится, потому что каждый следующий ход будет состоять в том, что игрок будет переворачивать верхнюю карту.

Написать программу, которая находит, после скольких ходов возникнет такая ситуация.

Входные данные. В первой строке текстового файла `tops.sis` дано количество карт в колоде N ($1 \leqslant N \leqslant 300$). Во второй строке дано N разделённых пробелами целых чисел A_i ($1 \leqslant A_i \leqslant N$; $A_i \neq A_j$, если $i \neq j$) — расположение карт в колоде сверху вниз.

Выходные данные. В единственную строку текстового файла `tops.val` вывести одно целое число — количество ходов, которые сделает игрок, пока не увидит сверху колоды единицу. Можно предполагать, что ни в одном тесте ответ не превысит 10 000.

Пример.

`tops.sis`	`tops.val`
`4` `3 2 4 1`	`2`

Начальное расположение карт `3 2 4 1`. На первом ходу игрок перевернёт первые три карты (`3 2 4` → `4 2 3`). Получится колода `4 2 3 1`. На втором ходу игрок перевернёт четыре карты (то есть всю колоду). В результате получится `1 3 2 4`, и больше в игре ничего меняться не будет.

4. Copy-Paste

1 секунда 40 очков

За свою шалость Юку получил от учителя наказание — сто раз написать фразу “Я должен хорошо себя вести.”

В целях экономии природных ресурсов учитель приказал написать это не на бумаге, а в компьютере.

Юку написал фразу один раз и обнаружил, что программа, которая есть в компьютере учителя, позволяет помимо простого введения текста ещё следующие операции: комбинация клавиш Ctrl-A выделяет весь текст; комбинация клавиш Ctrl-C копирует выделенный текст в буфер обмена, и комбинация клавиш Ctrl-V добавляет содержимое буфера обмена в текущий документ.

Если в момент нажатия Ctrl-V часть текста в документе выделена, то выделенная часть стирается, и содержимое буфера обмена вставляется вместо неё; если ничего не выделено, то содержимое буфера обмена просто добавляется в документ.

Конечно, Юку хочет написать фразу ровно столько раз, сколько требуется — не больше, ведь иначе учитель просто не поверит, что он случайно написал больше.

Написать программу, которая поможет Юку найти кратчайшую последовательность клавиш, с которой он сможет размножить одно предложение ровно столько раз, сколько требуется.

Входные данные. В единственной строке текстового файла `acvv.sis` дано число N ($1 \leqslant N \leqslant 1\,000\,000$): сколько раз нужно написать предложение.

Выходные данные. В единственную строку текстового файла `acvv.val` вывести одно число — минимальное число нажатии клавиш в результате которвх из одного предложения получится ровно N.

Пример.

acvv.sis	acvv.val
3	5

Ctrl-A выделяет весь текст, Ctrl-C копирует его в буфер, Ctrl-V замещает весь текст содержимым буфера обмена (в документе по-прежнему одно предложение), и ещё два нажатия Ctrl-V добавляют к тексту ещё два предложения.

5. Multi-Ant

1 секунда 50 очков

Компиляция и сборка большого программного продукта, состоящего из множества зависящих друг от друга модулей, может быть достаточно ресурсоёмкой работой. Ant — одна из программ для автоматизации этой работы.

Для запуска Ant необходимо задать все необходимые шаги и их зависимости. Ant выполнит все эти шаги в правильном порядке, то есть ни один шаг не начнётся, пока не завершатся все те шаги, результаты которых для него необходимы.

К сожалению, для больших проектов этого недостаточно, потому что выполнение всех необходимых шагов на одном компьютере просто-напросто занимает слишком много времени. Multi-Ant — это следующее поколение Ant, предназначенное для распределения необходимых шагов между несколькими компьютерами.

Написать компонент Multi-Ant, который получает список шагов, необходимых для сборки проекта, и зависимости между ними, и составляет план их выполнения на нескольких компьютерах так, чтобы проект собирался за минимальное время. На выполнение каждого шага уходит одна минута. Если два шага не зависят друг от друга, то их можно выполнять одновременно на разных компьютерах.

Входные данные. В первой строке текстового файла `mant.sis` дано число шагов N ($1 \leqslant N \leqslant 1000$), и в каждой из следующих N строк описаны зависимости очередного шага в виде *шаг зависимость1 зависимость2 ...*, где *шаг* — это имя самого шага, и *зависимость1*, *зависимость2*, ... — имена тех шагов, которые должны быть выполнены прежде, чем можно начинать шаг *шаг*. Имена шагов состоят из $1 \ldots 20$ маленьких латинских букв и разделены одним пробелом. Длина любой строки в файле не превышает 250 знаков.

Выходные данные. В первую строку текстового файла `mant.val` вывести время T, необходимое для выполнения всех шагов, и в следующие T строк вывести план действий по минутам. Во вторую строку файла вывести (разделённые пробелами) имена шагов, которые должны выполняться параллельно в первую минуту, в третью строку — шаги, которые должны выполняться во вторую минуту, и так далее. Можно предположить, что компьютеры имеются в неограниченном количестве — при необходимости фирма докупит ещё. Если планов с минимальным временем несколько, вывести любой из них.

Пример.

`mant.sis`

```
8
checkout clean
clean init
compile checkout clean
coverage compile
deploy test
init
report coverage
test compile
```

`mant.val`

```
6
init
clean
checkout
compile
test coverage
deploy report
```

6. Минотавр

50 очков

Как вы помните, архитектор Дедал с сыном Икаром пытались сбежать с острова Крит, улетев на крыльях. Но Икар взлетел слишком высоко в небо, так что солнце растопило воск, которым были прикреплены его крылья. Хотя легенда гласит, что это случилось над морем и Икар утонул, последние археологические исследования показали, что на самом деле он упал в лабиринт, построенный Дедалом для Минотавра — чудовища с телом человека и головой быка.

Пока Икар падает, ему сверху хорошо виден план лабиринта и положение Минотавра в нём. Поскольку крылья его уже почти не держат, у него остаётся очень мало времени, чтобы решить, куда падать.

Помоги Икару найти такое место для приземления, чтобы кратчайший путь Минотавра до места приземления Икара был максимальным.

План лабиринта можно рассматривать как поле из $N \times M$ квадратов, каждый из которых — либо стена, либо проход, и из одного прохода в другой можно попасть только в том случае, если у них есть общая грань.

Входные данные. В первой строке входного файла задан порядковый номер теста T, а также длина N и ширина M лабиринта. В каждой из следующих N строк дано ровно M знаков. Эти строки отображают план лабиринта, причём '`.`' означает проход, '`X`' — стену и '`M`' — положение Минотавра. Известно, что в лабиринте не больше одного входа-выхода.

Выходные данные. В первую строку выходного файла вывести порядковый номер теста T, и в каждую из следующих N строк вывести ровно M знаков. Эти N строк должны отображать точно тот же заданный план лабиринта, в котором ровно один символ '`.`' заменён символом '`I`' так, что если Икар приземлится в этот квадрат, то кратчайший путь Минотавра к этому квадрату будет максимальным. Если ответов с максимальной длиной несколько, вывести любой из них.

Пример.

Входной файл	Выходной файл

`0 5 5`
`XXXXX`
`X...X`
`XXX.X`
`XM..X`
`XXXXX`
	`0`
	`XXXXX`
	`XI..X`
	`XXX.X`
	`XM..X`
	`XXXXX`

Оценивание. В этом задании будет дано 10 комплектов входных данных в файлах от `minotest.01.sis` до `minotest.10.sis`, и в качестве решения необходимо представить соответствующие выходные файлы. Представлять программу не обязательно, она не оценивается.

Примечание. Ещё приложена вспомогательная программа для просмотра лабиринтов и вычисления расстояний. Программа дана в файле `MinoTool.jar`, для её нужна среда Java Runtime, которую можно получить по адресу `http://www.java.com/getjava/`.