Задача A. Окружности

Автор:	Юрий Петров
Имя входного файла:	`circles.in`
Имя выходного файла:	`circles.out`
Ограничение по времени:	3 секунды (*4 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Рассмотрим множество точек на координатной плоскости. Для каждой точки построена окружность радиуса R. Для каждой окружности найдите количество точек данного множества, лежащих на этой окружности.

Формат входных данных

Входные данные состоят из одного или более тестов.

Каждый тест начинается строкой, содержащей два целых числа: n — количество точек и R^2 — квадрат радиуса ($1 \leqslant n \leqslant 10^5$, $1 \leqslant R^2 \leqslant 10^9$). Следующие n строк содержат по два целых числа: координаты x_i, y_i точек ($0 \leqslant x_i, y_i \leqslant 10^9$). Гарантируется, что все точки различны.

Общее количество точек во входных данных не превысит 10^5. Общее количество тестов во входных данных не превышает 1000. Входные данные завершаются строкой, содержащей пару нулей, которая не является тестом.

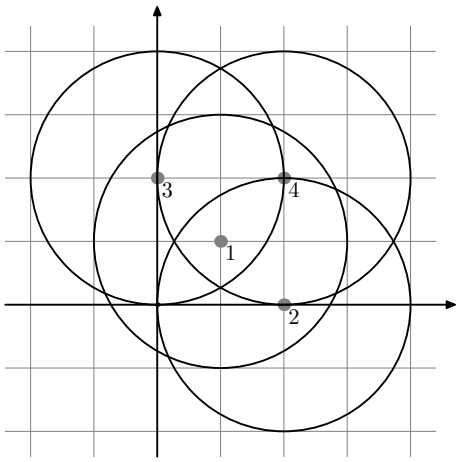
Формат выходных данных

В ответ на каждый тест выведите строку, содержащую n чисел: количество точек данного множества, лежащих на каждой из n заданных окружностей, в порядке следования их центров во входных данных.

Пример

circles.in	circles.out
4 4 1 1 2 0 0 2 2 2 0 0	0 1 1 2

Иллюстрация



Задача B. CRC (Division 1 Only!)

Автор:	Антон Майдель
Имя входного файла:	`crc.in`
Имя выходного файла:	`crc.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

В научно-исследовательском институте данных строк (НИИДС) хранится большой объём строк. Для каждой строки хранится также её циклический избыточный код (CRC). Жёсткие диски в серверах НИИДС настолько надёжны, что в каждой строке может быть испорчен не более чем один бит. Васе поручено написать программу, которая исправляет ошибки в строках. Для этого ему нужно решить следующую задачу.

Рассмотрим многочлены от одной переменной над полем $GF(2)$ (это означает, что коэффициенты могут принимать лишь значения 0 и 1, а все вычисления с ними производятся по модулю 2). Пусть $P(x) = x^{32} + x^{26} + x^{15} + x^7 + 1$. Многочлен $P(x)$ обладает следующим интересным свойством: для любых различных целых чисел i и j таких, что $0 \leqslant i, j < 2^{32}$, многочлены $x^i \bmod P(x)$ и $x^j \bmod P(x)$ также различны.

Здесь $A(x) \bmod B(x)$ означает остаток от деления многочлена $A(x)$ на многочлен $B(x)$. Формально, $A(x) \bmod B(x) = R(x)$, где максимальная степень вхождения x в $R(x)$ строго меньше, чем максимальная степень вхождения x в $B(x)$, а также существует многочлен $Q(x)$ такой, что $A(x) = Q(x) \times B(x) + R(x)$. Как и в случае целочисленного деления, для любого многочлена $A(x)$ и любого ненулевого многочлена $B(x)$ существуют единственные такие $Q(x)$ и $R(x)$, что равенство выше верно. К примеру, $x^{32} \bmod P(x) = x^{26} + x^{15} + x^7 + 1$ (напомним, что все коэффициенты вычисляются по модулю 2); здесь $Q(x) = 1$.

Задача Васи — для каждого заданного многочлена $S(x)$ найти минимальное неотрицательное целое k такое, что $x^k \bmod P(x)$ равно $S(x)$. Помогите ему это сделать.

Формат входных данных

Входные данные состоят из одного или более тестов.

Каждый тест состоит из одной строки, в которой задан многочлен $S(x)$. Каждый многочлен состоит из одного или более одночлена; соседние одночлены разделяются символом '+'. Каждый одночлен записывается как `x^k`, где показатель степени k — целое число такое, что $0 \leqslant k < 32$. Одночлены различны и записаны в порядке убывания k. Во входных данных нет пробелов.

Общее количество тестов во входных данных не превосходит 200. Входные данные завершаются строкой, содержащей символ '0', которая не является тестом.

Формат выходных данных

В ответ на каждый тест выведите одну строку, содержащую ответ на задачу.

Пример

crc.in	crc.out
x^0 x^1 x^26+x^15+x^7+x^0 x^26+x^21+x^15+x^13+x^7+x^6+x^0 x^31+x^25+x^14+x^6 0	0 1 32 38 4294967294

Задача C. Целые точки (Division 1 Only!)

Автор:	Иван Казменко
Имя входного файла:	`integerpoints.in`
Имя выходного файла:	`integerpoints.out`
Ограничение по времени:	3 секунды (*4 секунды для Java*)
Ограничение по памяти:	256 мебибайт

По трём данным целым числам p, q и n найдите количество точек на координатной плоскости, имеющих целые координаты и удовлетворяющих неравенствам

$$0 \leqslant x \leqslant n \text{ и } 0 \leqslant y \leqslant x \cdot \sqrt{\frac{p}{q}}.$$

Формат входных данных

На первой строке ввода задано целое число t — количество тестов ($1 \leqslant t \leqslant 100\,000$). За ней следует t строк, каждая из которых содержит один тест. Каждый тест состоит из трёх целых чисел p, q и n, разделённых одиночными пробелами ($1 \leqslant p, q \leqslant 100$, $0 \leqslant n \leqslant 10^9$).

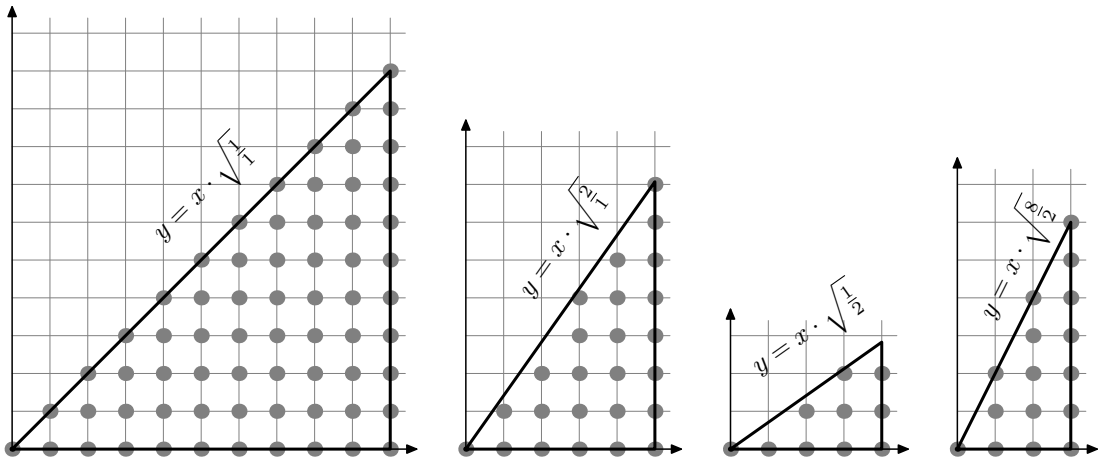
Формат выходных данных

В ответ на каждый тест выведите количество точек с целыми координатами, удовлетворяющих ограничениям, на отдельной строке.

Пример

integerpoints.in	integerpoints.out
4 1 1 10 2 1 5 1 2 4 8 2 3	66 25 10 16

Иллюстрация



Задача D. Игра на правильном многоугольнике

Автор:	Иван Казменко
Имя входного файла:	`ngon.in`
Имя выходного файла:	`ngon.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Правильный многоугольник — это многоугольник, у которого все стороны равны между собой и все углы равны между собой.

Рассмотрим правильный выпуклый многоугольник на плоскости с n вершинами. Каждая вершина либо свободна, либо отмечена. Перед началом игры все вершины свободны. Два игрока делают ходы по очереди. На каждом ходу игрок выбирает k свободных вершин, являющихся вершинами правильного выпуклого k-угольника, и отмечает их. Здесь k может быть любым делителем n таким, что $1 < k < n$. Особый случай возникает для $k = 2$: в этой игре правильным выпуклым 2-угольником считается пара диаметрально противоположных вершин исходного n-угольника. Правильных выпуклых 2-угольников нет, если n нечётно. Игрок, который не может сделать ход, проигрывает, а другой игрок объявляется победителем.

По данному целому числу n выясните, какой из игроков выиграет при правильной игре.

Формат входных данных

На первой строке ввода задано целое число t — количество тестов ($1 \leqslant t \leqslant 100$). За ней следует t строк, каждая из которых содержит один тест. Каждый тест состоит из одного целого числа n ($3 \leqslant n \leqslant 10^9$).

Формат выходных данных

В ответ на каждый тест выведите одну строку, содержащую одно слово. Это слово должно быть либо «`First`», если выигрывает первый игрок, либо «`Second`», если выигрывает второй игрок. Считайте, что оба игрока действуют оптимально.

Пример

ngon.in	ngon.out
2 4 6	Second First

Задача E. Числа

Автор:	Андрей Лопатин
Имя входного файла:	`numbers.in`
Имя выходного файла:	`numbers.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Вася знает, что $2+2\times2$ — не то же самое, что $(2+2)\times2$. У него есть массив A, состоящий из n целых чисел, и его задача очень проста: необходимо сказать, сколько можно найти различных троек (i, j, k) таких, что $A_i+A_j\times A_k = (A_i+A_j)\times A_k$. Здесь индексы i, j и k могут принимать все возможные значения от 1 до n; заметьте, что индексы могут совпадать. Две тройки индексов (i_1, j_1, k_1) и (i_2, j_2, k_2) считаются различными, если хотя бы одно из равенств $i_1 = i_2$, $j_1 = j_2$ и $k_1 = k_2$ неверно.

Помогите Васе найти количество таких троек.

Формат входных данных

На первой строке ввода задано целое число t — количество тестов ($1 \leqslant t \leqslant 10\,000$). Каждый тест состоит из двух строк. В первой из этих строк записано целое число n ($1 \leqslant n \leqslant 100\,000$), а вторая строка содержит n целых чисел A_i, не превосходящих 10^9 по модулю. Суммарная длина массивов во всех тестах не превосходит 100 000.

Формат выходных данных

Для каждого теста ваше решение должно вывести количество троек, подходящих под вышеописанное условие, на отдельной строке.

Пример

numbers.in	numbers.out
2 3 -1 0 1 6 -1 -1 0 0 1 1	15 120

Задача F. Пары чисел

Автор:	Юрий Петров
Имя входного файла:	`pairs.in`
Имя выходного файла:	`pairs.out`
Ограничение по времени:	5 секунд (*6 секунд для Java*)
Ограничение по памяти:	256 мебибайт

Для данного массива целых чисел $a_1, a_2, \ldots, a_n$ и целого положительного числа X найдите количество пар индексов (i, j) таких, что $i < j$ и $\gcd(|a_i - a_j|, X) = 1$.

Здесь $\gcd(p, q)$ обозначает наибольший общий делитель чисел p и q. Заметьте, что $\gcd(0, p) = p$ для любого целого положительного числа p.

Формат входных данных

Входные данные состоят из одного или более тестов.

Каждый тест начинается строкой, содержащей два целых числа: n, количество элементов массива, и X ($1 \leqslant n \leqslant 10^5$, $1 \leqslant X \leqslant 10^8$). Следующая строка содержит n целых чисел a_i ($1 \leqslant a_i \leqslant 10^9$).

Общая количество элементов во всех массивах во входных данных не превысит 10^5. Общее количество тестов во входных данных не превышает 1000. Входные данные завершаются строкой, содержащей пару нулей, которая не является тестом.

Формат выходных данных

В ответ на каждый тест выведите одну строку, содержащую число пар.

Пример

pairs.in	pairs.out
7 30 1 2 3 4 5 6 7 0 0	6

Задача G. Сжатие строки (Division 1 Only!)

Автор:	Антон Майдель
Имя входного файла:	`strpack.in`
Имя выходного файла:	`strpack.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Научно-исследовательский институт данных строк (НИИДС) дал Васе задание написать программу для сжатия строк.

Рассмотрим строку s над алфавитом из N букв, пронумерованных числами 1, 2, ..., N. Для каждой буквы i Васе известна частота f_i — сколько раз буква i встречается в строке s. Каждой букве i Вася должен сопоставить непустой код c_i, состоящий только из нулей и единиц. Для различных i и j код c_i не может оказаться префиксом кода c_j. Кроме того, длина каждого кода c_i не может превосходить заданное целое число L. Наконец, после замены в строке s каждой буквы i на её код c_i длина полученной закодированной строки должна быть минимально возможной при приведённых выше ограничениях.

Помогите Васе найти длины кодов c_i.

Формат входных данных

Входные данные состоят из одного или более тестов.

Каждый тест состоит из двух строк. В первой из них задано через пробел два целых положительных числа N и L ($N \leqslant 100$, $L \leqslant 30$). Вторая строка содержит N чисел f_i ($1 \leqslant f_i \leqslant 1\,000\,000\,000$); соседние числа разделены одним пробелом.

Общее количество тестов во входных данных не превосходит 500. Входные данные завершаются строкой, содержащей пару нулей, которая не является тестом.

Формат выходных данных

В ответ на каждый тест выведите две строки. В первой из этих строк выведите одно число — минимальную возможную длину закодированной строки при заданных ограничениях. Во второй строке выведите N чисел — длины кодов c_1, c_2, ..., c_N. Разделяйте соседние числа одним пробелом.

Гарантируется, что хотя бы одно решение существует. Если возможных решений несколько, можно выводить любое из них.

Пример

strpack.in	strpack.out
1 1 1 8 3 1 1 2 3 5 8 13 21 7 7 1 2 3 5 8 13 21 0 0	1 1 162 3 3 3 3 3 3 3 3 124 6 6 5 4 3 2 1

Задача H. Диаметр дерева (Division 1 Only!)

Автор:	Юрий Петров
Имя входного файла:	`treed.in`
Имя выходного файла:	`treed.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Бензопила «Дружба» может использоваться для деревьев до 2.5 метров в диаметре...

Чтобы сформулировать задачу, напомним некоторые определения из теории графов:

Расстоянием между вершинами v и u называется минимальное количество ребер в пути от v до u.
Эксцентриситетом вершины называется максимальное из расстояний от данной вершины до других вершин.
Диаметром графа называется максимальный эксцентриситет среди всех его вершин.

Рассмотрим невзвешенный неориентированный граф. Изначально граф состоит из одной вершины и не содержит рёбер. В граф последовательно добавляются вершины. Каждая вершина добавляется вместе с ребром, соединяющим её с некоторой другой вершиной, уже содержащейся в графе.

Ваша задача — определить диаметр графа после каждой операции.

Формат входных данных

Входные данные состоят из одного или более тестов.

Каждый тест начинается строкой, содержащей одно целое число n, количество вершин графа после всех операций ($2 \leqslant n \leqslant 10^5$). Следующая строка содержит $n-1$ целое число: a_2, a_3, …, a_n ($1 \leqslant a_i < i$). Эти числа определяют операции: вершина с номером i будет добавлена вместе с ребром (i, a_i). Вершина с номером 1 — та вершина, которая есть в графе изначально, остальные вершины нумеруются с 2 в том порядке, в котором они добавляются.

Общее число вершин во всех графах не превосходит 10^5. Входные данные завершаются строкой, содержащей один ноль, которая не является тестом.

Формат выходных данных

В ответ на каждый тест выведите одну строку, содержащую $n-1$ целое число: диаметр графа после каждой операции добавления. Разделяйте соседние числа в строке пробелами.

Пример

treed.in	treed.out
13 1 1 3 2 1 1 5 7 9 10 4 7 0	1 2 3 4 4 4 5 5 6 7 7 7

Задача I. Универсальный путь

Автор:	Иван Казменко
Имя входного файла:	`universalpath.in`
Имя выходного файла:	`universalpath.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Рассмотрим лабиринты размера 10×10 квадратных клеток. Каждая клетка либо пуста, либо заполнена непроходимой стеной. Одна из клеток лабиринта выбрана в качестве стартовой позиции, другая — в качестве пункта назначения.

После этого в стартовую позицию ставится Робот. Робот может делать шаги; один шаг перемещает Робота в соседнюю клетку в одном из четырёх основных направлений. Робот не может делать шаг в стену, а также покидать пределы лабиринта.

У Робота есть программа, которую он должен выполнить. Программа — это строка, состоящая из следующих команд: сделать один шаг на север (‘`N`’), на запад (‘`W`’), на юг (‘`S`’) или на восток (‘`E`’). Команды выполняются последовательно в том порядке, в котором они записаны в программе. Для каждой команды, если Робот может сделать шаг из своей текущей позиции в заданном направлении, он делает этот шаг. В противном случае (если в этом направлении находится стена, или же этот шаг выводит Робота за пределы лабиринта) Робот просто игнорирует команду.

Задача Робота — пройти лабиринт, то есть оказаться в пункте назначения. Если в какой-то момент Робот перемещается в пункт назначения, его задача выполнена. Если Робот выполнил все команды в своей программе, но ни разу не побывал в пункте назначения, его задача провалена.

Количественная мера достижений Робота — штраф, определяемый следующим образом. Если задача выполнена, штраф — это количество команд, выполненных перед тем, как Робот в первый раз побывал в пункте назначения (здесь считаются как команды, в результате которых был сделан шаг, так и команды, которые были проигнорированы). Если же задача Робота провалена, штраф равен константе 20 000.

Ваша задача — написать такую программу для Робота, чтобы у него хорошо получалось проходить лабиринты *в среднем*, то есть чтобы средний штраф не превысил заданный предел t. В вашей программе должно быть не более 10 000 команд.

Ваша программа будет проверяться следующим образом. Всего жюри заготовило 20 тестов. В каждом тесте, на вход вашему решению подаётся лишь целое число t. У жюри есть также 1000 заранее построенных лабиринтов для каждого теста (всего 20 000 лабиринтов), но они держатся в секрете. После того, как ваше решение выведет программу для Робота, на каждом тесте проверяющая программа запускает её на 1000 заранее построенных лабиринтов. Если среднее значение штрафа на этих лабиринтах не превысило заданный предел, ваше решение проходит этот тест; в противном случае оно получает вердикт «`Wrong Answer`» на этом тесте.

Все заранее построенные лабиринты сгенерированы следующим алгоритмом. Сначала на поле 10×10 случайным образом выбирается 20 различных клеток. После этого следует 500 итераций добавления стен. На каждой итерации выбирается случайная клетка на поле 10×10. Если эта клетка свободна и не отмечена, на неё ставится стена. После этого, если все отмеченные клетки всё ещё достижимы друг из друга, стена остаётся на клетке; в противном случае она убирается, и клетка опять становится свободной.

После того, как стены установлены, выбираются стартовая позиция и пункт назначения. Для каждой пары достижимых друг из друга клеток лабиринта расстоянием будем считать

длину кратчайшего пути между ними; здесь длина пути — это количество шагов в нём. Далее случайным образом выбирается одна из тех пар клеток, для которых расстояние получилось максимальным. Одна из клеток пары становится стартовой позицией, а другая — пунктом назначения.

В описанном выше алгоритме все случаи случайного выбора независимы и равномерно распределены. Все лабиринты генерируются случайно и независимо. Заметьте, что вашему решению не требуется хорошо работать на лабиринтах, отличных от тех, которые строятся по этому алгоритму.

Формат входных данных

На первой строке ввода задано целое число t — предел среднего штрафа ($1000 \leqslant t \leqslant 1500$). Всего в этой задаче 20 тестов. Гарантируется, что в тесте с номером n предел равен $t = \max(1000, 1550 - n \cdot 50)$. Так, предел равен 1500 в тесте 1, 1450 в тесте 2, 1400 в тесте 3, ..., 1050 в тесте 10 и 1000 в тестах 11–20. Первые 10 тестов могут показать, насколько хорошо работает ваша программа. Последние 10 тестов предназначены для проверки того, что ваша программа действительно хорошо работает.

Заметьте, что ваше решение не обязано использовать число, заданное во вводе, это число дано лишь для удобства.

Формат выходных данных

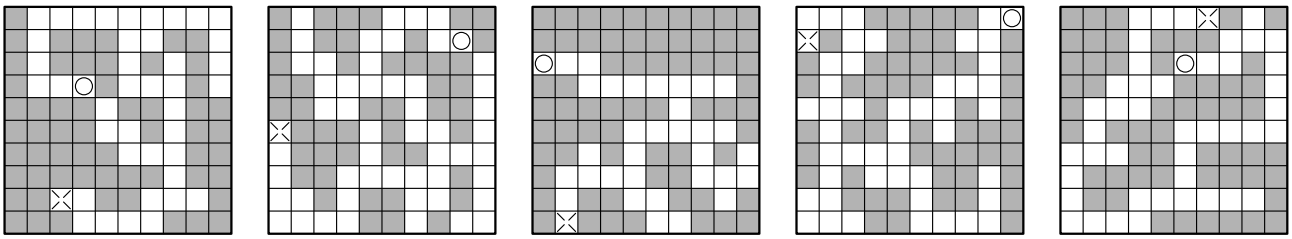
Ваше решение должно выводить программу для Робота на отдельной строке. Её длина должна быть не более 10 000 символов. Каждый символ должен быть либо '`N`' для движения на север, либо '`W`' для движения на запад, либо '`S`' для движения на юг, либо '`E`' для движения на восток.

Помните, что, если вашему решению требуется значительное время для нахождения хорошего ответа, вы можете найти этот ответ на своём локальном компьютере и послать лишь программу, которая его выводит.

Пример

В этой задаче нет примеров тестов.

В качестве иллюстрации к работе алгоритма, генерирующего лабиринты, ниже приводится несколько сгенерированных им лабиринтов.



Задача J. ISBN (Division 2 Only!)

Имя входного файла:	`isbn.in`
Имя выходного файла:	`isbn.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

International Standard Book Number (ISBN) — это 13-значное число, которое используется в качестве уникального идентификатора книги. При этом для защиты от ошибок при записи цифр ISBN имеет следующую особенность: если умножить первую цифру числа на 1, вторую на 3, затем третью снова на 1 и так далее, после чего полученные произведения сложить, то сумма будет делиться на 10.

Например, для числа 8790525616955 имеем

$8 \cdot 1 + 7 \cdot 3 + 9 \cdot 1 + 0 \cdot 3 + 5 \cdot 1 + 2 \cdot 3 + 5 \cdot 1 + 6 \cdot 3 + 1 \cdot 1 + 6 \cdot 3 + 9 \cdot 1 + 5 \cdot 3 + 5 \cdot 1$

Подсчитывая, получаем, что сумма равна 120.

Напишите программу, которая по 13-значному числу определяет, может ли это число быть корректным ISBN.

Формат входных данных

Входной файл содержит одно 13-значное целое число без ведущих нулей.

Формат выходных данных

Выведите “`Yes`”, если число во входном файле может быть использовано в качестве корректного ISBN, и “`No`” в противном случае.

Примеры

isbn.in	isbn.out
8790525616955	Yes
1111111111111	No

Задача K. Из префиксной в постфиксную (Division 2 Only!)

Имя входного файла:	`postfix.in`
Имя выходного файла:	`postfix.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Префиксная запись является одним из альтернативных способов записи арифметических выражений. При обычном способе записи выражений, иначе называемый инфиксной записи, бинарный оператор размещается между операндами, например, `4 + 3`, а в префиксной записи оператор располагается перед операндами. Приведённый выше пример запишется, следовательно, как `+ 4 3`. Аналогично, префиксная запись для $3 - 2$ выглядит как `- 3 2`.

Одним из преимуществ префиксной системы записи выражений является тот факт, что можно обходиться без скобок. Например, $6-(4-1)$ записывается как `- 6 - 4 1`, а $(6-4)-1$ — как `- - 6 4 1`. Иначе префиксную запись называют польской системой записи, так как она была предложена польским логиком Яном Лукашевичем в 1920 году.

В постфиксной, или обратной польской, записи оператор размещается после операндов. Например, постфиксная запись инфиксного выражения $(6 - 4) - 1$ выглядит как `6 4 - 1-`.

Вам задано выражение, записанное в префиксной записи. Требуется перевести его в постфиксную запись.

Формат входных данных

Входной файл состоит из не более, чем 100 тестовых примеров. Каждый тестовый пример записан в отдельной строке и представляет собой арифметическое выражение, записанное в префиксной нотации. Выражение может содержать целые числа от 0 до 9 включительно, а также операторы ‘+’ и ‘-’. Соседние объекты (операторы или цифры) отделены друг от друга одиночными пробелами. Завершающая строка входного файла содержит одно число 0, которое обрабатывать не нужно. Гарантируется, что во всех тестовых примерах префиксное выражение непусто, корректно и содержит менее 20 операторов.

Формат выходных данных

Для каждого выражения во входном файле выведите в отдельной строке это же выражение в постфиксной записи. Соседние объекты отделяйте одиночными пробелами.

Пример

postfix.in	postfix.out
1 - - 6 4 1 0	1 6 4 - 1-

Задача L. Перфоратор (Division 2 Only!)

Имя входного файла:	`punchy.in`
Имя выходного файла:	`punchy.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

В начале компьютерной эры для записи программ использовались перфокарты. Каждый оператор занимал одну перфокарту. Перфокарты складывались в стопку, после чего загружались в считыватель, обрабатывающий их снизу вверх. Так что порядок перфокарт был очень важен. Перфокарты перевязывали ленточкой или клали в специальную коробку, чтобы предотвратить их перемешивание.

Однажды Билл оставил колоду перфокарт около открытого окна, так что ветер разнёс перфокарты по всей комнате. Билл решил провести эксперимент, собрал колоду в случайном порядке и отправил на выполнение.

Ваша задача — проэмулировать исполнение этой программы.

Формат входных данных

Язык программирования, на котором была написана программа, использует ровно две переменные (A и B) и семь различных операторов.

Перед началом выполнения программы переменные A и B равны нулю.

Каждая строка входного файла содержит один оператор. Каждый оператор начинается с целого числа в интервале от 1 до 7 включительно, за которым, возможно, следует имя переменной, а за ним, возможно, число или имя переменной.

В нижеприведённом описании операторов X или Y обозначает произвольную переменную (A или B).

- `1 X n` — присвоить переменной X целое значение n;
- `2 X` — вывести на экран значение переменной X;
- `3 X Y` — вычислить $X + Y$ и запомнить результат в переменной X;
- `4 X Y` — вычислить $X \cdot Y$ и запомнить результат в переменной X;
- `5 X Y` — вычислить $X - Y$ и запомнить результат в переменной X;
- `6 X Y` — вычислить частное целочисленного деления X/Y и запомнить результат в переменной X.
- `7` — остановить выполнение программы.

Гарантируется, что при выполнении деления никогда не происходит деления на 0, и что при выполнении всех остальных операторов вычисляемые или присваиваемые значения являются целыми числами, не превосходящими по модулю 10^4.

(Пояснение; при делении отрицательных чисел $-3/2$ и $3/-2$ дают частное -1, а $-3/-2$ — частное 1.)

Формат выходных данных

Ваша программа должна выполнить программу, при этом вывод каждого оператора типа 2 идёт с новой строки. После того, как на вход поступит оператор типа 7, программа должна завершить работу.

Пример

punchy.in	punchy.out
1 A 3 1 B 4 2 B 2 A 3 A B 2 A 5 A A 2 A 2 B 7	4 3 7 0 4

Задача M. Глобальное потепление (Division 2 Only!)

Имя входного файла:	`warming.in`
Имя выходного файла:	`warming.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Одна из гипотез, объясняющих изменение средней температуры, заключается в том, что в течение длительных периодов времени средняя температура меняется в некоторой циклической закономерности, при этом каждый новый цикл начинается с более высокого значения температуры. Средняя температура берётся на пятигодичных промежутках, и выражается в десятых долях градуса.

Так, например, если последовательность наблюдений выглядит следующим образом:

`3, 4, 6, 4, 5, 7, 5`

то можно заметить, что температура сначала увеличилась на одну десятую, затем увеличилась на две, затем уменьшилась на две, затем увеличилась на одну, затем увеличилась на две и затем снова уменьшилась на две десятых. Таким образом, существует цикл длины 3, который описывает изменения температуры: $(+1, +2, -2)$. Иначе говоря, если мы посмотрим на разности величин, начиная с начала последовательности, то последовательность разностей представляется в виде двукратно повторённого фрагмента $(+1, +2, -2)$.

Другой пример: пусть мы имеем следующую последовательность наблюдений:

`3, 4, 6, 7.`

В этом случае изменение температуры выглядит так: на одну десятую вверх, на две десятые вверх, на одну десятую вверх. В этом случае считаем, что минимальный цикл имеет длину 2 $(+1, +2)$. Этот цикл за время наблюдений проходит один раз полностью и один раз частично.

Вам задана последовательность наблюдений. Ваша задача — найти кратчайший цикл, который бы соответствовал данным наблюдениям.

Формат входных данных

Входной файл состоит из нескольких тестовых примеров (не более пяти). Каждый тестовый пример начинается с целого числа n ($1 \leqslant n \leqslant 20$), задающего количество наблюдений, за которым следует n целых чисел, по модулю не превосходящих 1000 — наблюдавшиеся средние температуры. Входной файл завершается строкой, состоящей из одного нуля. Эту строку обрабатывать не требуется.

Формат выходных данных

Для каждого тестового примера выведите в отдельной строке длину наименьшего цикла. Отметим, что цикл существует всегда — например, можно рассмотреть всю последовательность как один большой цикл.

Пример

warming.in	warming.out
7 3 4 6 4 5 7 5 3 1 3 5 3 1 4 5 4 3 4 6 7 0	3 1 2 2