

Администрация города Нижнего Новгорода  
Департамент образования и социально-правовой защиты детства

Нижегородский государственный университет  
им. Н. И. Лобачевского

Институт прикладной физики РАН  
Научно-образовательный центр

# Нижегородские городские олимпиады школьников по информатике

2005 — 2010

Третье издание, дополненное

Под общей редакцией В. Д. Лелюха

Нижний Новгород  
ИПФ РАН  
2010

**Нижегородские** городские олимпиады школьников по информатике: 2005 — 2010 / Под общ. ред. В. Д. Лелюха.— 3-е изд., дополн.— Н. Новгород: ИПФ РАН, 2010.— 144 с.

В сборнике представлены задачи, предлагавшиеся на Нижегородских городских олимпиадах школьников по информатике с 2005 по 2010 г.; каждая задача сопровождается решением и примером программы. Это задачи на разные темы (в том числе и математические). Здесь и простые арифметические представления, и комбинаторные задачи, решаемые перебором. Но есть и геометрия, причём иногда использующая для построения эффективного метода решения необычные для школьной программы представления (двулистная плоскость в задаче о плавании на яхте). По возможности, объяснения, предлагаемые в разборах решений, подчёркнуто используют естественные, простые для школьника интерпретации. Все задачи могли бы быть решены и иными способами (в рамках тех же учебных программ), но программирование их в большинстве случаев приводило бы или к медленным, или к сложным в реализации алгоритмов.

Рекомендуется учителям информатики, а также школьникам (и студентам), готовящим себя к участию в олимпиадах по программированию.

УДК 004.42(079.1.063)

Рецензент И. А. Шерешевский

Результаты, архивы и другие материалы олимпиад можно найти на сайте <http://olympiads.nnov.ru>

Электронную версию книги также можно найти на сайте <http://olympiads.nnov.ru>

Оригинал-макет подготовлен в системе  $\text{\LaTeX} 2_{\epsilon}$

# Содержание

<b>Введение</b> . . . . .	5
Нижегородские городские олимпиады школьников по информатике . . . . .	5
Задачи олимпиадной информатики . . . . .	7
<b>I олимпиада (19 февраля 2005 г.)</b> . . . . .	12
1. Билеты . . . . .	12
2. Детали . . . . .	14
3. Числа . . . . .	16
4. Период дроби . . . . .	18
5. Разрезание многоугольника . . . . .	20
6. Перекрасить поле . . . . .	23
Результаты . . . . .	26
<b>II олимпиада (27 декабря 2005 г.)</b> . . . . .	29
1. Счастливые билеты . . . . .	29
2. Сломанный калькулятор . . . . .	33
3. Принтер . . . . .	35
4. Сортировка . . . . .	38
5. Мах . . . . .	40
6. Клеточное море . . . . .	42
Результаты . . . . .	46
<b>III олимпиада (30 января 2007 г.)</b> . . . . .	49
1. Бассейны и трубы . . . . .	49
2. Батон . . . . .	52
3. Обфускатор . . . . .	55
4. А ты купи коня! . . . . .	60
5. Строительство в городе . . . . .	64
6. Кофе . . . . .	68
Результаты . . . . .	70
<b>IV олимпиада (28 января 2008 г.)</b> . . . . .	72
1. Счастливые билеты — 2 . . . . .	72
2. Коды городов . . . . .	74
3. Собрать квадрат . . . . .	79
4. Мёртвый код . . . . .	81
5. Домофон . . . . .	84

---

6. Кривой горизонт . . . . .	89
Результаты . . . . .	94
<b>V олимпиада (22 января 2009 г.) . . . . .</b>	<b>96</b>
1. Счастливые билеты — 3 . . . . .	96
2. Олимпийская система — 2 . . . . .	99
3. Автостанции . . . . .	101
4. Протокол матча . . . . .	106
5. Дуга, отрезок и точка . . . . .	110
6. Списывание . . . . .	113
Результаты . . . . .	116
<b>VI олимпиада (4 февраля 2010 г.) . . . . .</b>	<b>119</b>
1. Метро . . . . .	119
2. Автомат со сдачей . . . . .	123
3. Логарифм . . . . .	126
4. Шарик . . . . .	128
5. Совершенно секретно . . . . .	131
6. Шифр . . . . .	134
Результаты . . . . .	140
<b>Список литературы . . . . .</b>	<b>142</b>

# Введение

## Нижегородские городские олимпиады школьников по информатике

До 1999—2000 учебного года Нижегородские городские олимпиады школьников по информатике проводились на базе Нижегородского государственного технического университета и входили в систему всероссийских олимпиад школьников. С 2000—2001 уч. г. олимпиада в течение нескольких лет не проводилась, и была восстановлена как отдельная олимпиада в 2004—2005 уч. г. Первая (восстановленная) олимпиада была проведена 19 февраля 2005 г., и с тех пор проведение городских олимпиад по информатике в Нижнем Новгороде вновь вошло в традицию.

В олимпиаде каждый год принимают участие около 70 учащихся 6—11 классов из Нижнего Новгорода, а начиная со второй олимпиады — и из Нижегородской области (из Сарова и Балахны); от 20 до 40 из них награждаются дипломами. Для обеспечения такого количества участников компьютерными местами тур олимпиады проводится одновременно на нескольких площадках: в Научно-образовательном центре Института прикладной физики РАН (НОЦ ИПФ РАН), в Нижегородском государственном университете им. Н. И. Лобачевского, в лицее 40, в лицее 8 и в школе 30. Участникам предлагается в течение 4,5—5 часов решить 6 задач различного уровня сложности. После окончания тура участники, сопровождающие и учителя перемещаются в конференц-зал НОЦ ИПФ РАН, где решения школьников проверяются автоматической проверяющей системой с выводом подробных результатов на экран. По окончании тестирования происходит подведение итогов олимпиады и награждение победителей. На закрытии олимпиады всем присутствующим раздаются книжки с условиями задач, подробными решениями и примерами правильных программ.

Организаторами олимпиады являются департамент образования и социально-правовой защиты детства администрации города Нижнего Новгорода и Нижегородский государственный университет им. Н. И. Лобачевского. Спонсорами олимпиады выступают ведущие IT-компании Нижнего Новгорода: компания Intel — Нижний Новгород и компания

Мега; они обеспечивают питание во время тура и после его окончания, а также предоставляют ценные призы победителям.

В этой книге приведены задачи со всех восстановленных олимпиад, т. е. начиная с 2005 г. К каждой задаче приведены решение и пример правильной программы. За основу были взяты решения из книжек, раздававшихся участникам на закрытиях соответствующих олимпиад; во многих случаях эти разборы были доработаны или добавлены альтернативные методы решения. Многие решения описаны с таких позиций, что для их понимания не требуется дополнительных знаний в области алгоритмов. В некоторых задачах может потребоваться или оказаться полезным знание различных основных понятий алгоритмического программирования и простейших алгоритмов. В качестве введения в эту тему можно рекомендовать книгу [1] или классический многотомник [2], а также книгу [3].

Примеры программ приведены для дополнительной иллюстрации способов реализации изложенного алгоритма. При подготовке каждой задачи членами жюри олимпиады были написаны несколько программ-решений, зачастую на разных языках программирования,— их можно скачать с официального сайта олимпиады <http://olympiads.nnov.ru>. Там же можно скачать реализации альтернативных методов решения, упоминаемых в разборах задач.

Эту книгу в электронном виде, а также её исходный код (для системы ЛАТЭХ) также можно скачать с сайта <http://olympiads.nnov.ru>.

## Задачи олимпиадной информатики

В этом разделе кратко описано, какого типа задачи бывают на олимпиадах по информатике, что требуется от участника, чтобы решить их, и как обычно проходит проверка решений. Описание соответствует наиболее часто встречающимся видам задач; иногда бывают задачи и правила их проверки, немного отличающиеся от приведённых ниже, подобные случаи обычно явно оговариваются в правилах олимпиады.

Задачи олимпиад по информатике, как правило, составляются так, чтобы дать участникам возможность проявить свои навыки в составлении *алгоритмов* решения задачи и в реализации выбранного алгоритма в программном коде без ошибок. Многие технические проблемы, имеющие место в реальном промышленном программировании (например, создание удобного пользовательского интерфейса), намеренно исключаются, чтобы позволить участникам сосредоточиться на алгоритме.

Поэтому в олимпиадах по информатике, а также во многих аналогичных соревнованиях, сложился традиционный формат задач. В большинстве случаев по каждой задаче участники должны написать программу, которая будет принимать некоторые входные данные и решать описанную в условиях задачу для этих данных. Входные данные программа должна считывать из определённого файла (иногда с клавиатуры); результаты решения (ответ на задачу) программа также должна выводить в определённый файл (иногда на экран); никакого другого функционала от программы не требуется. Время допустимой работы программы на одном входном файле (одном тесте), а также допустимый объём используемой памяти обычно строго ограничены, что позволяет отделить эффективные решения задачи от неэффективных.<sup>1</sup>

---

<sup>1</sup> Явное ограничение объёма используемой памяти для каждой задачи имеет смысл лишь при использовании современных 32-битных компиляторов. На старых 16-битных компиляторах под DOS (Borland Pascal, Borland C и т. п.) общий объём памяти, к которой может получить доступ программа без использования особых приёмов, автоматически ограничен общим объёмом свободной нижней памяти, который обычно составляет порядка 450–550 Кб и должен указываться в правилах олимпиады. Поэтому при проведении олимпиады на старых компиляторах ограничение по памяти обычно специально не указывается. Все нижегородские городские олимпиады, представленные в этом сборнике, в силу ряда причин проводились на 16-битных компиляторах под DOS, и потому ограничение памяти во всех задачах не указано.

Задачи, приведённые в этом сборнике, можно, конечно, решать и на современных компиляторах, если помнить об ограничениях памяти старых компиляторов. При

Такой формат задач позволяет писать решения практически на любом современном языке программирования общего назначения. Как правило, на олимпиаде участникам предоставляется на выбор несколько широко распространённых языков и сред программирования, и каждый участник может использовать наиболее удобные из них.

Условие задачи обычно содержит пять частей:

- Заголовок задачи, как правило указывающий её номер, название, имена входного и выходного файла, ограничение по времени и по памяти, и максимальный балл, который даётся за полное решение этой задачи;
- Основной текст, содержащий постановку задачи;
- Формат входных данных, описывающий, что программа должна ожидать во входном файле, а также указывающий ограничения на размер входных данных и другие ограничения, которые могут иметь значение при разработке алгоритма;
- Формат выходных данных, описывающий, что и в каком формате программа должна вывести в выходной файл;
- Пример входного файла и правильного соответствующего ему выходного файла (при наличии нескольких возможных правильных ответов могут существовать и другие правильные выходные файлы, которые не указаны в примере).

По каждой задаче участник, как правило, должен сдать одну программу-решение (в виде исходного кода). Программы-решения проверяются по принципу «чёрного ящика»: единственным критерием, по которому оцениваются программы, считается правильность создаваемых программой выходных файлов для корректных (т. е. удовлетворяющих формату входных данных) входных файлов при условии соблюдения ограничений по времени и памяти. Соответственно, полным решением задачи считается программа, которая для любого входного файла, удовлетворяющего формату входных данных и ограничениям, указанным там, формирует правильный выходной файл. Критерии правильности выходного файла определяются задачей и обычно указываются в тек-

---

необходимости установки строгих ограничений памяти для современных компиляторов, как правило, можно установить разумные значения, используемые в большинстве современных олимпиад (порядка десятков–сотен мегабайт), при этом все задачи, конечно, будут иметь решения, и нередко даже технически несколько более простые за счёт того, что не потребуются специфическая техника работы с динамической памятью и т.п. Тем не менее, в отдельных случаях (в первую очередь, в задаче «Коды городов» с четвёртой олимпиады) появятся новые, существенно более простые идейно способы решения, что очень упростит задачу.

сте условий; одним из критериев всегда является строгое соблюдение формата выходного файла.

В частности, в решениях участников не оценивается ни стиль написания программы (расстановка отступов, выбор имён переменных, наличие комментариев и т.д.), ни конкретный способ реализации выбранного алгоритма (при условии, что выбранный способ укладывается в ограничения); это связано с невозможностью строгой формализации и объективизации подобной оценки, а также с тем, что отчасти указанные особенности программы являются техническими и не требуются для демонстрации эффективности и корректности выбранного алгоритма. Кроме того, программы не должны проверять корректность входных данных, поскольку эта задача тоже, как правило, является технической и не представляет сложности с точки зрения выбора алгоритма решения. Участники не должны также предоставлять жюри какое-либо описание алгоритма или доказательство его корректности помимо правильно работающей программы.

Сказанное в предыдущем абзаце не обозначает, что участникам не нужно доказывать корректность выбранного алгоритма или соблюдать элементарные требования стиля при написании программы, поскольку всё это напрямую влияет на правильность программы. Действительно, участник может быть уверен в корректности своей программы, только если он доказал правильность алгоритма; а удобное форматирование кода и выбор простой реализации очень упростят ему проверку написанного кода и поиск ошибок в нем.

Благодаря принципу «чёрного ящика» проверка решений становится максимально строгой и объективной. Более того, появляется возможность автоматизации этого процесса, в результате чего на всех современных олимпиадах непосредственно проверка решений участников осуществляется автоматически (программно) с минимальным участием человека. (В отличие от олимпиад по другим точным предметам, таким как математика или физика.)

В процессе подготовки олимпиады жюри для каждой задачи формирует набор тестов — набор входных файлов, максимальной полно покрывающий всё множество допустимых входных файлов и неправильных решений и позволяющий корректно оценить как полные, так и частичные решения задач; количество тестов обычно варьируется от десяти до ста. (В большинстве задач проверка решений участников путём проверки их программ на *всех допустимых* входных файлах невозможна в силу огромного количества последних, поэтому проводится про-

верка лишь на очень небольшом их подмножестве.) По каждому тесту также подготавливается «файл ответа» — правильный выходной файл для этого теста. Кроме того, по каждой задаче пишется специальная программа («проверяющая программа»), которая принимает на вход три файла: входной файл, файл ответа и некоторый выходной файл, — и оценивает правильность последнего. В простейших случаях, когда ответ на задачу абсолютно однозначен, проверяющая программа ограничивается сравнением выходного файла с файлом ответа; в более сложных случаях написание проверяющей программы может само по себе представлять нетривиальную задачу.

После этого проверка решений участников осуществляется специальной оболочкой — «тестирующей системой», которая для каждого решения участника и для каждого теста по соответствующей задаче запускает это решение на этом тесте, после чего запускает проверяющую программу этой задачи, которая определяет правильность ответа, выданного программой участника. Этот процесс называется тестированием, поскольку состоит в запуске программы-решения по очереди на различных тестовых примерах. Отметим, что он не имеет ничего общего с тестированием как выбором из нескольких данных вариантов одного правильного.

Итогом проверки является определение по каждому тесту результата прохождения этого теста. Как правило, это может быть:

- Тест успешно пройден;
- Превышение ограничения на время работы или на память;
- Несоблюдение формата выходных данных (т. е. проверяющая программа не смогла разобрать выходной файл);
- Неправильный ответ (т. е. выходной файл сформирован корректно, но не является правильным выходным файлом для этого теста; например, найдено неоптимальное решение и т.п.).

Крайне важным является строгое соблюдение формальных требований к программе: следование формату выходных данных, использование правильных имён файлов и т. д. Поскольку программы проверяются автоматически, любая ошибка может привести к тому, что тест не будет зачтён. Отметим, что соблюдение этих требований не так сложно и требует от участника в первую очередь внимательности.

На школьных олимпиадах, как правило, участники в течение всего тура работают над задачами, и их решения проверяются лишь после окончания тура. При проверке за каждый успешно решённый (пройденный) тест начисляется определённое количество баллов так, что сумма

баллов за все тесты по одной задаче равна заявленному заранее общему баллу за эту задачу; иногда встречаются более сложные алгоритмы начисления баллов. Существуют и другие форматы олимпиад, отличающиеся методикой оценки решений; например, во многих олимпиадах (в первую очередь, в командных олимпиадах) участники могут сдавать решения на проверку в течение тура и получать от жюри ответ о результатах сдачи, а при подведении окончательных итогов учитывается время, потраченное участниками на решение каждой задачи.

Отметим, что многие олимпиадные задачи, особенно высокого уровня (уровня областных, всероссийских и международных олимпиад) требуют достаточно серьезной теоретической подготовки в так называемой области *computer science* (включающей различные алгоритмы и методы), которая не всегда обеспечивается школьной программой (в отличие от, например, олимпиад по математике или физике). В качестве литературы по олимпиадным задачам можно рекомендовать книгу [4], содержащую большой набор задач с подробными решениями и описаниями общих методов, а также литературу, цитированную там. Основные алгоритмы и приёмы решения задач также хорошо изложены в книге [1].

# I городская олимпиада школьников по информатике 19 февраля 2005 г.

## Авторы задач

Демидов А. Н., студент 4 курса ВМК ННГУ;  
 Калинин П. А., студент 3 курса ВШОПФ ННГУ;  
 Круглов А. А., студент 5 курса ВШОПФ ННГУ;  
 Лелюх В. Д., старший преподаватель ННГУ;  
 Политов С. В., студент 4 курса мехмата ННГУ;  
 Тимушев Р. И., студент 2 курса ВШОПФ ННГУ;  
 Хаймович И. М., студент 2 курса ВШОПФ ННГУ.

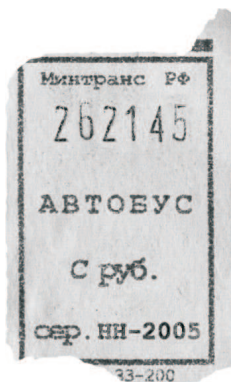
## Задача 1. Билеты

<i>Входной файл</i>	tickets.in
<i>Выходной файл</i>	tickets.out
<i>Ограничение по времени</i>	2 с
<i>Максимальный балл за задачу</i>	100

Ваш друг работает кондуктором в маршрутке. Очень часто он сталкивается со следующей проблемой: ему передают некоторую сумму денег и забывают сообщить, сколько билетов надо. В результате ему приходится гадать, сколько же билетов хотели купить.

Недавно он сделал одно очень интересное наблюдение: как правило, человек, передавая за проезд, не даёт лишних купюр. Т. е., если пассажир подготовил некоторый набор купюр, чтобы купить несколько билетов, но это же количество билетов можно купить, передав только некоторые купюры (не все) из этого набора, то он не будет передавать такой набор. Например, чтобы купить один билет стоимостью 6 рублей, человек не будет передавать две купюры по 10 рублей (т. к. достаточно и одной десятки).

С другой стороны, пассажиры, как правило, считать умеют, и переданных денег обычно достаточно, чтобы купить нужное число билетов.



Напишите программу, которая, зная, какой набор денег передали кондуктору, и считая, что выполняются эти два предположения (что не дают лишних денег и что денег хватает), определит, сколько билетов мог хотеть купить пассажир. Гарантируется, что хотя бы одно подходящее количество билетов существует.

### Формат входных данных

В первой строке входного файла находятся два целых числа  $N$  и  $c$  — общее количество переданных купюр и стоимость одного билета соответственно ( $1 \leq N \leq 1\,000$ ,  $1 \leq c \leq 100\,000$ ).

Во второй строке входного файла находятся  $N$  целых чисел — номиналы переданных купюр (каждый номинал находится в пределах от 1 до 100 000 включительно).

### Формат выходных данных

В выходной файл выведите два целых числа — сначала минимальное количество билетов, которое могло быть нужно пассажиру, потом — максимальное.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
2 6 10 10	2 3

### Решение

Фактически, в данной задаче требуется определить все количества билетов  $k$  такие, что их стоимость  $ck$  меньше суммарной стоимости всех переданных купюр, но больше суммарной стоимости любого подмножества этого набора, не совпадающего с полным набором.

Определим, какие условия на  $k$  накладывают эти два требования.

Пусть  $s$  — суммарная стоимость всех переданных купюр; тогда очевидно, что первое условие равносильно условию  $k \leq s/c$ .

Пусть  $s_1$  — суммарная стоимость некоторого подмножества купюр, не совпадающего с полным набором. Тогда для выполнения второго условия необходимо, чтобы было  $k > s_1/c$ . Пусть  $m$  — наименьшая стоимость среди переданных купюр, тогда, очевидно,  $s - m$  есть наибольшая суммарная стоимость среди всех подмножеств купюр, не совпадающих с полным набором. Следовательно, второе условие равносильно условию  $k > (s - m)/c$ .

Так как  $k$  должно быть целым числом, то эти два условия превращаются в систему

$$\begin{cases} k \leq \lfloor \frac{s}{c} \rfloor \\ k > \lfloor \frac{s-m}{c} \rfloor, \end{cases}$$

где  $\lfloor x \rfloor$  — целая часть  $x$ .

Последнее условие равносильно условию

$$k \geq \left\lfloor \frac{s-m}{c} \right\rfloor + 1,$$

поэтому ответом на задачу являются числа

$$\left\lfloor \frac{s-m}{c} \right\rfloor + 1 \quad \text{и} \quad \left\lfloor \frac{s}{c} \right\rfloor.$$

## Пример правильной программы

```

var f:text;
    n:integer;
    s,min,c,a:longint;
    i:integer;
begin
  assign(f,'tickets.in');reset(f);
  read(f,n,c);
  s:=0;
  min:=1000000;
  for i:=1 to n do begin
      read(f,a);
      s:=s+a;
      if a<min then
        min:=a;
      end;
      assign(f,'tickets.out');rewrite(f);
      writeln(f,(s-min) div c+1,' ');
      s div c);
      close(f);
    end.

```

## Задача 2. Детали

*Входной файл*

*Выходной файл*

*Ограничение по времени*

*Максимальный балл за задачу*

details.in

details.out

2 с

100

A	A		B	B	B
A					B
A	A	A			B
A			B	B	B

На клеточном поле  $N \times M$  расположены две жёсткие детали. Деталь  $A$  накрывает в каждой строке несколько (не ноль) первых клеток, деталь  $B$  — несколько (не ноль) последних; каждая клетка либо полностью накрыта одной из деталей, либо нет.

Деталь  $B$  начинают двигать влево, не поворачивая, пока она не упрётся в  $A$  хотя бы одной клеткой. Определите, на сколько клеток будет сдвинута деталь  $B$ .

## Формат входных данных

В первой строке входного файла расположены два числа  $N$  и  $M$  — число строк и столбцов соответственно ( $1 \leq N, M \leq 100$ ). Далее следуют  $N$  строк, задающих расположение деталей. В каждой находится ровно  $M$  символов 'A' (клетка, накрытая деталью A), 'B' (накрытая деталью B) или '.' (свободная клетка).

## Формат выходных данных

Выведите в выходной файл одно число — ответ на задачу.

## Пример

<i>Входной файл</i>	<i>Выходной файл</i>
<pre>4 6 AA.BVV A...B AAA..B A..BVB</pre>	<pre>1</pre>

*Примечание:* Пример соответствует рисунку.

## Решение

Это самая простая задача олимпиады, рассчитанная на то, что с ней по силам справиться каждому из участников.

Подсчитаем, сколько пустых клеток находится между двумя деталями в каждой строке. Из всех строк выберем ту, в которой это количество минимально (или одну из них, если таковых несколько). Покажем, что число пустот в этой строке и будет ответом на задачу.

Если мы сдвинули деталь B на одну клетку влево, то количество пустот в каждой строке уменьшится на единицу. Деталь упрётся лишь в том случае, когда в какой-то строке пустоты вообще исчезнут, а одной из первых таких строк будет именно выбранная. Осталось заметить, что, так как за каждый сдвиг число пустот в выбранной строке уменьшалось на единицу и в конце оно стало равным нулю, то количество сдвигов и есть начальное количество пустот в строке.

## Пример правильной программы

```
const inf = 'details.in';
      ouf = 'details.out';
var y, x:integer;
    c :char;
    i, j:integer;
    p :integer;

min :integer;
begin
  assign(input, inf);
  reset(input);
  assign(output, ouf);
  rewrite(output);
```

```

readln(y,x);
min:=101;
for i:=1 to y do begin
  p:=0;
  for j:=1 to x do begin
    read(c);
    if c='.' then
      inc(p);
  end;
end;

```

---

```

if p<min then
  min:=p;
readln;
end;
writeln(min);
close(output);
close(input);
end.

```

---

## Задача 3. Числа

<i>Входной файл</i>	numbers.in
<i>Выходной файл</i>	numbers.out
<i>Ограничение по времени</i>	2 с
<i>Максимальный балл за задачу</i>	100

Дана последовательность чисел  $a_1, a_2, \dots, a_N$ . За одну операцию разрешается удалить любое (кроме крайних) число, заплатив за это штраф, равный произведению этого числа на сумму соседних. Требуется удалить все числа, кроме крайних, с минимальным суммарным штрафом.

Например:

Начальная последовательность:

**1 50 51 50 1.**

Удаляем четвёртое число, штраф  $50 \cdot (1 + 51) = 2600$ , получаем

**1 50 51 1.**

Удаляем третье число, штраф  $51 \cdot (50 + 1) = 2601$ , получаем

**1 50 1.**

Удаляем второе число, штраф  $50 \cdot (1 + 1) = 100$ .

Итого штраф 5301.

### Формат входных данных

В первой строке входного файла расположено одно число  $N$  ( $1 \leq N \leq 100$ ) — количество чисел в последовательности.

Во второй строке находятся  $N$  целых чисел  $a_1, a_2, \dots, a_N$ ; никакое из чисел не превосходит по модулю 100.

### Формат выходных данных

Выведите в выходной файл одно число — минимальный суммарный штраф.

## Пример

Входной файл	Выходной файл
5 1 50 51 50 1	5301

## Решение

Задача решается методом динамического программирования.

Заведём квадратную матрицу  $M$  размера  $N \times N$ . В ячейке  $M_{ij}$  будем хранить минимальный штраф за удаление всех чисел с  $i$ -го по  $j$ -е не включительно (элементы матрицы, у которых  $i \geq j$ , нам не будут нужны). Подсчитывать эти штрафы будем следующим образом: поскольку среди чисел между  $i$ -м и  $j$ -м какое-то ( $k$ -е) мы должны удалить последним, то переберём все  $k$  от  $i$  до  $j$  не включительно и посчитаем наименьший штраф за удаление всех чисел между  $i$  и  $j$  при условии, что  $k$ -е удаляется последним. Этот штраф равен сумме наименьших штрафов за удаление всех чисел с  $i$  по  $k$  ( $M_{ik}$ ), с  $k$  по  $j$  ( $M_{kj}$ ) и штрафа за последнее удаление —  $(a_i + a_j) \cdot a_k$ . Тогда очевидно, что

$$M_{ij} = \min_{i < k < j} (M_{ik} + M_{kj} + a_k \cdot (a_i + a_j)).$$

Несложно посчитать элементы матрицы с индексами, отличающимися на 1: так как удалять между соседними числами нечего, то  $M_{i,i+1} = 0$ .

Осталось только вычислить все элементы матрицы (например, в порядке увеличения разности между индексами) и посмотреть на число  $M_{1N}$ , которое и будет ответом на задачу.

Кроме того, надо аккуратно обработать случай  $N = 1$ . Вышеприведённый алгоритм для этого случая не подходит, но очевидно, что, так как удалять при  $N = 1$  нечего, то ответ здесь — 0.

## Пример правильной программы

```
const inf=100*100*200*10;
var n:integer;
a:array[1..100] of integer;
M:array[1..100,1..100] of longint;
f:text;
i,l,k:integer;
min:longint;

begin
  assign(f,'numbers.in');
  reset(f);
  read(f,n);
```

```
for i:=1 to n do
  read(f,a[i]);
for i:=1 to n-1 do
  M[i,i+1]:=0;
for l:=3 to n do
  for i:=1 to n-l+1 do begin
    min:=inf;
    for k:=i+1 to i+l-2 do
      if min>M[i,k]+M[k,i+1-1]+
        a[k]*(a[i]+a[i+1-1]) then
        min:=M[i,k]+M[k,i+1-1]+
          a[k]*(a[i]+a[i+1-1]);
```

```

M[i,i+1-1]:=min;
end;
assign(f,'numbers.out');
rewrite(f);
if n<>1 then

```

```

writeln(f,M[1,n])
else writeln(f,0);
close(f);
end.

```

## Задача 4. Период дроби

<i>Входной файл</i>	period.in
<i>Выходной файл</i>	period.out
<i>Ограничение по времени</i>	2 с
<i>Максимальный балл за задачу</i>	100

Рассмотрим дробь  $1/n$ ,  $n \geq 2$ . Как известно, цифры в её десятичной записи начиная с некоторого места повторяются. Минимальную по длине повторяющуюся (без промежутков) часть называют периодом; часть после запятой, которая не входит ни в один период, назовём предпериодом.

Говоря строго, пусть

$$\frac{1}{n} = 0,a_1a_2a_3\dots,$$

причём запись не кончается на все девятки, т. е. для всех  $n_0 \geq 1$  существует  $n \geq n_0$  такое, что  $a_n \neq 9$ .

Как известно, для любого  $n \geq 2$  существует такая пара целых чисел  $(m, l)$ ,  $m > 0$ ,  $l \geq 0$ , что  $a_j = a_{j+m}$  для всех  $j > l$ . Выберем среди всех таких пар пары с наименьшим  $m$ , а среди них — пару с наименьшим  $l$ . Тогда  $m$  называется *длиной периода*, а  $l$  — *длиной предпериода* дроби  $1/n$ ; строка  $'a_{l+1}a_{l+2}\dots a_{l+m}'$  называется её *периодом*, а  $'a_1a_2\dots a_l'$  — *предпериодом*.

Например,

$$n = 28:$$

$1/28 = 0,03(571428)$ , предпериод '03', его длина 2; период '571428', длина 6;

$$n = 5:$$

$$1/5 = 0,2(0), \text{ предпериод '2', длина 1; период '0', длина 1.}$$

$$n = 3:$$

$$1/3 = 0,(3), \text{ предпериод пустой (''), длина 0; период '3', длина 1.}$$

По данному числу  $n$  найдите длину предпериода и длину периода у дроби  $1/n$ .

## Формат входных данных

Во входном файле находится одно число  $n$  ( $2 \leq n \leq 1\,000\,000$ ).

## Формат выходных данных

В выходном файле должны находиться два числа — сначала длина предпериода, потом длина периода дроби  $1/n$ .

## Пример

<i>Входной файл</i>	<i>Выходной файл</i>
28	2 6
5	1 1
3	0 1

## Решение

Для нахождения длины периода можно использовать известный алгоритм, основанный на поиске повтора в последовательности остатков, получаемых при делении столбиком. Однако для нахождения предпериода этот алгоритм может потребовать большое количество памяти для хранения уже встречавшихся остатков. Можно этого избежать, если заметить, что длина предпериода равна максимальной степени 2 или 5, на которую делится  $n$ . Докажем это. Заодно мы получим дополнительное упрощение алгоритма, поскольку нам удастся вообще избавиться от наличия непустого предпериода у  $1/n$ .

Дробь  $1/n$  может быть записана в виде

$$\frac{1}{n} = 0, \overline{a_1 \dots a_l (b_1 \dots b_p)},$$

где  $l$  — длина предпериода,  $p$  — длина периода.

Очевидно, длиной периода  $p$  будет минимальное натуральное число  $p'$ , для которого дробь

$$\frac{10^{p'} - 1}{n} = \overline{0, a_1 \dots a_l (b_1 \dots b_p)} \cdot 10^{p'} - \overline{0, a_1 \dots a_l (b_1 \dots b_p)}$$

обрывается, т. е. оканчивается на все нули. При этом длиной предпериода будет номер позиции последней различающей цифры в записи чисел  $\overline{0, a_1 \dots a_l (b_1 \dots b_p)} \cdot 10^p$  и  $\overline{0, a_1 \dots a_l (b_1 \dots b_p)}$ , т. е.

$$\frac{10^p - 1}{n} = \overline{0, a_1 \dots a_l (b_1 \dots b_p)} \cdot 10^p - \overline{0, a_1 \dots a_l (b_1 \dots b_p)} = \dots c_0, c_1 \dots c_l,$$

причём  $c_l \neq 0$ ,  $l$  — длина предпериода. Таким образом, мы доказали, что длина предпериода  $l \geq 0$  и длина периода  $p \geq 1$  — минимальные числа, для которых  $10^l(10^p - 1)$  делится на  $n$  (записывается как  $10^l(10^p - 1) : n$ ). Если переписать  $n$  в виде  $n = 2^{n_2} 5^{n_5} n'$ , где  $n'$  не делится ни на 2, ни на 5, получим  $10^l : 2^{n_2} 5^{n_5}$  и  $10^p - 1 : n'$ , или, по-другому,

$$l = \max(n_2, n_5).$$

Из  $10^p - 1 : n'$  получаем, что длина периода для  $1/n$  совпадает с длиной периода для  $1/n'$ , причём у  $1/n'$  период начинается сразу после запятой, поскольку  $n'$  не делится ни на 2, ни на 5.

### Пример правильной программы

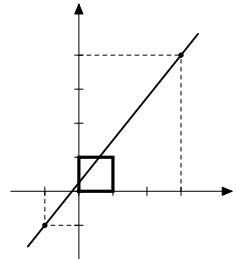
<pre>var   n:longint;   l,p:longint;  procedure solve; var n2,n5:integer;     k:longint; begin   n2:=0;   while n mod 2=0 do begin     n:=n div 2;     inc(n2);   end;   n5:=0;   while n mod 5=0 do begin     n:=n div 5;     inc(n5);   end;   if n2&gt;n5 then l:=n2   else l:=n5;</pre>	<pre>p:=1; k:=10 mod n; {всегда будет k=10^p (mod n)} while (k&lt;&gt;1)and(k&lt;&gt;0) do begin   k:=(k*10) mod n;   inc(p); end; {10^p=1 (mod n)} end;  begin   assign(input,'period.in');   reset(input);   readln(n);   solve;   assign(output,'period.out');   rewrite(output);   writeln(l,' ',p);   close(output);   close(input); end.</pre>
---	--

## Задача 5. Разрезание многоугольника

<i>Входной файл</i>	cut.in
<i>Выходной файл</i>	cut.out
<i>Ограничение по времени</i>	2 с
<i>Максимальный балл за задачу</i>	100

На плоскости задан многоугольник и прямая; прямая не проходит через вершины многоугольника (см. пример на рисунке). Напишите программу, которая определит, на сколько частей делит эта прямая многоугольник.

Если ваша программа будет правильно решать задачу для случая вертикальной прямой, то она наберёт не менее 30 баллов.



### Формат входных данных

В первой строке входного файла находится одно число  $N$  ( $3 \leq N \leq 1025$ ) — число вершин многоугольника. Далее следуют  $N$  строк, задающие вершины многоугольника: в  $i$ -й из этих строк находятся два целых числа  $x_i$  и  $y_i$  ( $|x_i|, |y_i| \leq 10\,000$ ) — координаты  $i$ -й вершины многоугольника. Вершины занумерованы от 1 до  $N$  в порядке обхода против часовой стрелки.

В  $(N + 2)$ -й строке входного файла находятся четыре целых числа  $p_x, p_y, q_x, q_y$ , задающие прямую (все числа не превосходят по модулю 10 000). Прямая проходит через точки  $(p_x, p_y)$  и  $(q_x, q_y)$ ; эти точки различны.

### Формат выходных данных

В выходной файл выведите одно число — ответ на задачу.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
4 0 0 1 0 1 1 0 1 -1 -1 3 4	2

*Примечание:* Пример соответствует рисунку.

### Решение

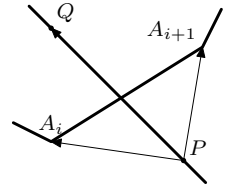
В этой задаче требовалось найти количество частей, на которые прямая разделяет многоугольник.

Очевидным образом количество точек пересечения прямой и границы многоугольника чётно. Действительно, ни одна вершина, а следовательно ни одна сторона многоугольника, не лежит на секущей прямой. Кроме того, если мы будем двигаться по прямой из бесконечности, то сколько раз мы «войдём» в многоугольник, то есть пересечём его границу, проходя извне внутрь, столько же раз и «выйдем», то есть пересечём его границу, проходя изнутри вовне.

Ясно, что каждая пара «вход—выход» в/из многоугольника увеличивает количество частей на 1.

Так как сначала часть была одна — сам многоугольник, то окончательная формула для количества частей следующая:  $(\text{количество пересечений})/2 + 1$ .

Теперь посчитаем количество пересечений. Проще всего для каждого ребра многоугольника проверить, лежат ли его концы по разные стороны от прямой. Вершины  $A_i(x_i; y_i)$  и  $A_{i+1}(x_{i+1}; y_{i+1})$  многоугольника лежат по разные стороны от прямой  $P(p_x; p_y)Q(q_x; q_y)$  тогда и только тогда, когда векторные произведения векторов  $\overrightarrow{PA_i}\{x_i - p_x; y_i - p_y\}$  и  $\overrightarrow{PA_{i+1}}\{x_{i+1} - p_x; y_{i+1} - p_y\}$  на вектор  $\overrightarrow{PQ}\{q_x - p_x; q_y - p_y\}$  имеют разные знаки (см. рисунок):



$$\left[ \overrightarrow{PA_i} \times \overrightarrow{PQ} \right] \cdot \left[ \overrightarrow{PA_{i+1}} \times \overrightarrow{PQ} \right] < 0.$$

Возможной ошибкой является проверка того, что векторные произведения имеют разные знаки, непосредственно сравнением их произведения с нулем. Дело в том, что каждое векторное произведение по модулю может достигать 800 000 000, то есть их произведение в Pascal'e уже не влезет в `longint`, а в C — в `long`. В данном случае лучше записать более длинное условие: либо первое произведение меньше нуля, а второе больше, либо наоборот.

Другой способ состоит в том, чтобы вместо векторных произведений перемножать их знаки, как и сделано в примере программы.

## Пример правильной программы

```
const nm=1025;
var n,i:integer;
    x,y:array[1..nm+1] of longint;
    x1,y1,x2,y2:longint;
    int:longint;

function vects(x1,y1,
               x2,y2:longint):integer;
var vect:longint;
begin
    vect:=x1*y2-x2*y1;
    if vect>0 then
        vects:=1
    else vects:=-1;
end;

begin
    assign(input,'cut.in');
    reset(input);
    assign(output,'cut.out');
```

```
rewrite(output);
read(n);
for i:=1 to n do begin
    read(x[i],y[i]);
end;
x[n+1]:=x[1];
y[n+1]:=y[1];
read(x1,y1,x2,y2);
int:=0;
for i:=1 to n do begin
    if vects(x2-x1,y2-y1,
            x[i]-x1,y[i]-y1)*
       vects(x2-x1,y2-y1,
            x[i+1]-x1,y[i+1]-y1)<0 then
        inc(int);
end;
writeln((int div 2)+1);
close(input);
close(output);
end.
```

## Задача 6. Перекрасить поле

Входной файл

repaint.in

Выходной файл

repaint.out

Ограничение по времени

2 с

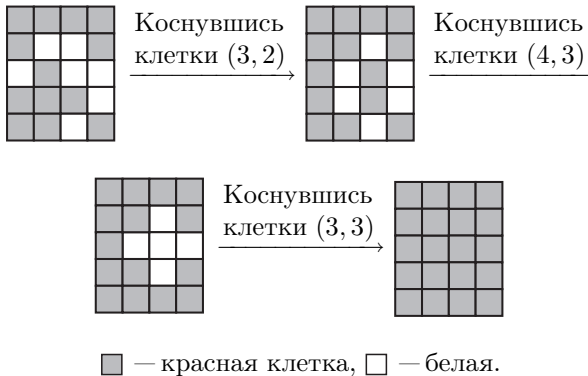
Максимальный балл за задачу

100

Рассмотрим следующую игру для одного игрока, проводимую на клеточном поле  $M \times N$ . Каждая клетка поля покрашена либо в красный, либо в белый цвет. За один ход можно коснуться одной из клеток волшебной палочкой, тогда эта клетка и все клетки, соседние с ней по стороне, сменят цвет на противоположный. Необходимо перекрасить все клетки в красный цвет за минимальное число ходов.

Строки поля нумеруются от 1 до  $M$  сверху вниз, столбцы — от 1 до  $N$  слева направо; в координатах клетки сначала задаётся номер строки, потом — номер столбца. Таким образом,  $(1, 1)$  — это левый верхний угол,  $(1, N)$  — правый верхний, а  $(M, N)$  — правый нижний.

Например,



### Формат входных данных

В первой строке входного файла находятся два числа  $M$  и  $N$  — число строк и число столбцов поля соответственно ( $1 \leq M \leq 500$ ,  $1 \leq N \leq 15$ ). Далее следуют  $M$  строк, задающих начальное состояние поля. В каждой из них находится ровно  $N$  символов:  $j$ -й символ в  $i$ -й из этих строк равен 'W', если клетка  $(i, j)$  белая, и 'R', если красная.

## Формат выходных данных

В первую строку выходного файла выведите одно число  $K$  — минимальное число ходов. В следующих  $K$  строках выведите ходы: в  $i$ -ю строку выведите два числа — номер строки и номер столбца клетки, которой надо коснуться палочкой на  $i$ -м ходу.

Если поле невозможно перекрасить, выведите в выходной файл одну строку 'NO' (без кавычек). Если существует несколько решений с минимальным числом ходов, выведите любое из них.

## Пример

<i>Входной файл</i>	<i>Выходной файл</i>
5 4	3
RRRR	3 2
RWWR	4 3
WRWW	3 3
RRRW	
RRWR	

*Примечание:* Пример соответствует рисунку.

## Решение

Для успешного решения этой задачи нужно было заметить следующий факт: пусть мы каким-либо способом определили, каких клеток надо коснуться в первой строке — тогда однозначно определяются клетки, которых надо коснуться во второй строке. Действительно, после того, как мы коснулись выбранных клеток первой строки, во второй строке мы обязаны коснуться тех клеток, над которыми в первой строке оказались белые клетки, и не можем касаться остальных. Но тогда мы аналогичным образом определяем, каких клеток надо коснуться в третьей строке, в четвёртой и т. д.

Таким образом, коснувшись некоторых клеток в первой строке, мы однозначно задаём касания клеток остальных строк. Когда мы дойдём до последней строки и коснёмся её клеток так, чтобы перекрасить предпоследнюю строку в красный цвет, возможны два варианта: в последней строке остались белые клетки, и, следовательно, такой вариант заполнения первой строки нас не устраивает, или последняя строка тоже стала вся красной, и у нас появился претендент на ответ задачи.

Перебрав все возможные способы касания клеток первой строки, и отобрав среди «хороших» вариантов тот, в котором суммарное число касаний было наименьшим, мы получим ответ.

При реализации этого алгоритма необходимо обратить внимание на быстрый подсчёт числа касаний. Для этого строки поля надо было хранить в виде одного машинного слова, биты в котором обозначают цвет клетки, а весь пересчёт при переходе на следующую строку выполнять операциями побитового сдвига и «исключающего или».

## Пример правильной программы

```

const infile = 'repaint.in';
      outfile = 'repaint.out';
      MaxM = 15;
      MaxN = 500;
      inf = 65535;

var i,j,n,m:integer;
    all,r,v,v1,v2,minr:word;
    min,count:word;
    a:array[0..MaxN] of word;
    ones:array[0..255] of byte;
    { ones[i] есть число единиц }
    { в битовой записи числа i }
    ch:char;

procedure print(c,r:word);
var v,v1,v2:word;
    i,j:integer;
begin
  writeln(c);
  v1:=r; v2:=0;
  for i:=0 to n-1 do begin
    v:=a[i] xor v1;
    for j:=0 to m-1 do
      if (v and (1 shl j))>0 then
        writeln(i+1,' ',j+1);
    v1:=((v shr 1) xor (v shl 1)
      xor v xor v2) and all;
    v2:=v;
  end;
end;

begin
  assign(input,infile);
  reset(input);
  assign(output,outfile);
  rewrite(output);
  read(n,m);
  all:=1 shl m-1;
  fillchar(a,sizeof(a),0);
  min:=inf;
  for i:=1 to n do
    for j:=0 to m-1 do begin
      repeat read(ch)
        until ch in ['W','R'];
      if ch='W' then
        a[i]:=a[i] or (1 shl j);
      end;
    for i:=0 to 255 do begin
      ones[i]:=0;
      for j:=0 to 7 do
        if (i and (1 shl j))>0 then
          inc(ones[i]);
      end;
    a[0]:=0;
    for r:=0 to all do begin
      count:=0;
      v1:=r;
      v2:=0;
      for i:=0 to n-1 do begin
        v:=a[i] xor v1;
        count:=count+ones[v and $FF]
          +ones[(v shr 8)];
        v1:=((v shr 1) xor (v shl 1)
          xor v xor v2) and all;
        v2:=v;
      end;
      if (a[n] xor v1=0)
        and(count<min) then begin
        min:=count;
        minr:=r;
      end;
    end;
  end;
  if min<inf then print(min,minr)
  else writeln('NO');
  close(output);
  close(input);
end.

```

## Результаты I Нижегородской городской олимпиады школьников по информатике

	Школа	Кл.	1	2	3	4	5	6	=	Дипл.
1. P39: Разенштейн Илья	лиц. 40	9	100	100	17	100	100	17	434	I
2. P34: Муравьев Александр	лиц. 40	10	100	76	16	100	100	13	405	I
3. P20: Песков Александр	165	10	100	100	38	52	100	0	390	I
4. P24: Смирнов Александр	165	11	85	100	26	48	100	13	372	I
5. P13: Вдовин Валерий	36	11	100	100	38	17	100	13	368	I
6. P32: Бударагин Дмитрий	лиц. 40	10	41	100	24	100	100	0	365	I
7. P33: Мишенин Александр	лиц. 40	10	100	100	16	58	82	6	362	I
8. P35: Парохоняк Анастасия	лиц. 40	10	53	100	6	39	100		298	II
9. P21: Голубев Кирилл	165	10	31	100	36	80	24	13	284	II
10. P14: Шмелев Александр	36	11	41	100	20	80	21	20	282	II
11. P25: Низовцев Сергей	165	7	100	100	9		72		281	II
11. P53: Лазарев Евгений	117	11	38	100	10	33	100	0	281	II
13. P22: Багиев Александр	165	10	33	100	17	66	35	13	264	II
14. P05: Склярков Олег	лиц. 38	10	31	100	21		100	0	252	II
14. P23: Лапшин Антон	165	11	10	100	27	15	100		252	II
16. P37: Рамин Виктор	лиц. 40	10	100	100	21	5	12		238	III
17. P72: Кажяев Дмитрий	11	11	33	100	9	5	90	0	237	III
18. P67: Галанин Александр	180	11	31	63	9	100	13	13	229	III
19. P12: Тураев Тимур	36	11	100	100	2	5	1	13	221	III
20. P38: Елифанов Владислав	лиц. 40	7	100	0	10	88	13	6	217	III
21. P50: Бражкин Дмитрий	82	11	33	100	8	20	41	13	215	III
22. P55: Зудин Илья	85	11	13	100	3	66	29		211	III
23. P62: Солюянов Михаил	14	11	41	100	10	5	41	13	210	III
24. P36: Путилов Алексей	лиц. 40	10	85	0	20	0	100		205	III
25. P52: Лопаткин Михаил	117	11	50	100		33	15		198	III
26. P63: Жегалин Николай	гимн. 80	11	90	100					190	III
27. P07: Боцьев Александр	лиц. 38	11	31	100	13	13	13	13	183	III
28. P76: Чистяков Андрей	32	7	80	100					180	
29. P09: Сасько Сергей	лиц. 38	11	41	100			33		174	
30. P06: Шумилов Вячеслав	лиц. 38	10	32	100	33	5			170	
30. P15: Тураев Марат	36	9	47	100	10		13		170	

32.	P18: Мхитарян Сергей	63	10	25	100	0	3	41	0	169	
32.	P48: Ершов Андрей	183	9	32	77	27		20	13	169	
34.	P40: Бенсон Янис	103	10	54	42	1	0	68	0	165	
35.	P51: Смирнова Ольга	82	11	13	100	21	11	9	6	160	
36.	P73: Иголкин Андрей	45	11	70	57	10	0	10	12	159	
37.	P19: Кузнецов Александр	105	10	46	100	6	5	1	0	158	
38.	P04: Клишин Алексей	лиц. 38	10	33	100	21		1		155	
38.	P10: Лукьяненко Илья	186	10	53	100	2		0		155	
40.	P78: Мойсеев Александр	17	10	46	100	4		3		153	
41.	P08: Миколайчук Николай	лиц. 38	11	16	100	12	19	0		147	
42.	P70: Бизяев Дмитрий	182	11	58	82			5		145	
43.	P61: Кольцов Максим	40	11	21	100	3	5	0	14	143	
44.	P81: Громов Кирилл	176	11	33	86	20	3			142	
45.	P27: Татарский Дмитрий			31	100	10	0			141	
46.	P68: Прытков Юрий	180	11	3	100	31	5	1		140	
46.	P77: Щербатовский Андрей	32	10	31	93	3	13		0	140	
48.	P17: Матросов Михаил	63	10	58	59	3	5			125	
49.	P44: Захаров Андрей	74	11	18	100	3		0		121	
50.	P01: Бовыкин Максим	лиц. 38	10	33	85	2				120	
51.	P79: Кочуков Максим	гимн. 2	11	11	100	2	2		0	115	
52.	P66: Каратаев Денис	гимн. 80	10	13	90	1				104	
53.	P69: Ким Евгений	94	11	0	100	0				100	
54.	P46: Бобылев Дмитрий	лиц. 87	9	13	79					92	
55.	P47: Крылова Ирина	лиц. 87	7	24	52	10			2	88	
56.	P11: Лысенков Илья	29	11	21	0	0	61	1	0	83	
57.	P02: Бронников Вячеслав	лиц. 38	10	31	42	0		1		74	
57.	P71: Малышев Георгий	11	11	11	30	3	28		2	74	
59.	P49: Сопин Дмитрий	82	10	19	0	6	5	12	13	55	
60.	P74: Волков Станислав	48	11	0	40	2	5			47	

В таблице не указаны 13 участников, набравшие менее 20 баллов.



# II городская олимпиада школьников по информатике 27 декабря 2005 г.

## Авторы задач

Демидов А. Н., студент 5 курса ВМК ННГУ;  
Калинин П. А., студент 4 курса ВШОПФ ННГУ;  
Круглов А. А., студент 6 курса ВШОПФ ННГУ;  
Лелюх В. Д., старший преподаватель ННГУ;  
Тимушев Р. И., студент 3 курса ВШОПФ ННГУ;  
Хаймович И. М., студент 3 курса ВШОПФ ННГУ.

## Задача 1. Счастливые билеты

<i>Входной файл</i>	ticket.in
<i>Выходной файл</i>	ticket.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Как известно, некоторые люди верят, что некоторые билеты являются счастливыми. Некоторые люди даже их коллекционируют. Эти люди нередко просят кондукторов продать им счастливый билет, а если им не продадут его, то иногда пытаются обменять билет на счастливый у других пассажиров.

Как известно, билеты у кондукторов занумерованы по порядку, поэтому они далеко не всегда могут удовлетворить такие просьбы пассажиров. Более того, обменять также получается не всегда, т. к. иногда среди большого количества последовательных билетов не бывает ни одного счастливого (например, все билеты от 998 999 по 999 998 являются несчастливыми).

Поэтому иногда пассажир, получив свой билет, хочет узнать, насколько близок его билет к счастливому. Некоторые пассажиры за решением этой задачи обращаются к кондукторам, в частности, к вашему другу, который по-прежнему работает кондуктором в маршрутке. Он, пытаясь решить эту задачу, понял, что самому ему не справиться, и, помня, как хорошо вы помогли ему в прошлый раз, опять обратился за

помощью к вам. Поскольку ваш друг считает, что не за горами введение восьмизначных номеров билетов, он хочет, чтобы ваша программа могла обрабатывать номера, состоящие из любого чётного количества цифр.

Поэтому вам предстоит написать программу, которая по заданному номеру билета найдёт счастливый билет, номер которого состоит из того же количества цифр и отличается от номера данного билета как можно меньше. Говоря строго, вам задан номер билета  $X$ , состоящий из  $2N$  цифр. Ваша программа должна найти такой счастливый номер билета  $Y$ , состоящий тоже из  $2N$  цифр, что модуль разности  $|X - Y|$  имеет наименьшее возможное значение.

Билет с номером из  $2N$  цифр (включая ведущие нули) назовём счастливым, если сумма первых  $N$  цифр его номера совпадает с суммой последних  $N$  цифр.

### Формат входных данных

В первой строке входного файла указан номер  $X$  билета. Длина номера чётна и находится в пределах от 2 до 200 включительно. Номер может содержать ведущие нули.

### Формат выходных данных

В первую строку выходного файла выведите номер счастливого билета, наиболее близкого к заданному. Этот номер должен состоять из такого же количества цифр, как и  $X$ , и так же может содержать ведущие нули.

В случае, когда решений несколько, можно выбрать любое из них.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
10	11
01234567	01234200
262145	262145
10000029999999	10000030000004
0001	0000

### Решение

Решение задачи могло быть таким: разобьём задачу на две. Найдём сначала ближайший счастливый билет, следующий за данным (или равный ему), затем предшествующий ему (или равный) и выберем из них ближайший.

Для решения первой части задачи (нахождения следующего счастливого билета) зафиксируем первые  $2N - k$  цифр (длина билета  $2N$ ), то есть оставим «свободными» последние  $k$  цифр. Будем считать, что точно меняется  $(2N - k + 1)$ -я цифра, и попробуем найти ближайший такой счастливый билет. Увеличим эту цифру на 1, а все последующие разряды занулим. Если эта цифра была 9, то мы не сможем прибавить 1 без изменения старших разрядов, поэтому в таком случае сразу переходим к следующему  $k$ . Далее, пока сумма цифр первой половины билета превышает сумму цифр второй половины не менее, чем на 9, будем, начиная с младших разрядов, добавлять 9 к соответствующему разряду. То есть начнём с самого младшего разряда: прибавим 9 в него. Если младший разряд не был нулевым (т. е.  $k \leq 1$ ), то мы не сможем этого сделать, и придётся перейти к следующему  $k$ . Если же предыдущая операция успешна, то прибавим 9 во второй разряд и т. д. После таких операций либо мы не сможем добавить очередную девятку (не хватит нулевых разрядов), либо суммы станут отличаться менее, чем на 9. В первом случае наша попытка с данным  $k$  не удалась, перейдём к следующему  $k$ . Во втором случае добавим (если сумеем) в следующий разряд разность сумм первой половины и второй половины билета. Если мы не сможем этого сделать (сумма цифры в данном разряде и добавляемой величины превысит 9, то есть будет перенос, что соответствует другому  $k$ ), то наша попытка при данном  $k$  неуспешна. Если же смогли, то мы нашли ближайший счастливый билет, следующий за данным, и дальнейшие поиски при больших  $k$  не дадут нужного результата (ближайшим он уже не будет). Будем идти с  $k = 0$  до  $k = 2N$ .  $k = 0$  соответствует проверке, что данный билет счастливый,  $k = 2N$  соответствует тому, что мы увеличили на 1 первую цифру билета (если эта цифра была 9, то мы уже нашли счастливый билет ранее, так как есть счастливый билет из всех девяток) и полностью переделали вторую половину. Таким образом, при  $k = 2N$  решение точно найдётся (хотя бы равное изменённой в первом разряде первой половине).

Для решения второй части задачи (нахождения предыдущего счастливого билета) произведём те же операции с фиксацией  $2N - k$  цифр. Отличия заключаются лишь в том, что из этого разряда мы вычитаем 1, а не прибавляем (если полученное число меньше 0, то попытка с данным  $k$  неуспешна, и надо переходить к следующему  $k$ ), во все последующие разряды кладём 9. Пока сумма цифр первой половины меньше суммы цифр второй половины более, чем на 9, будем отнимать 9, начиная с младших разрядов: сначала из младшего вычтем 9, затем из второго

и т. д. Аналогично первой части задачи разбираемся с последним оставшимся разрядом. Аналогично будем идти с  $k = 0$  до  $k = 2N$ .

## Пример правильной программы

```

type tlong=array[1..200] of integer;
var f:text;
    ff:file;
    x,y1,y2,d1,d2:tlong;
    n:integer;
    i,j:integer;
    ds:integer;
    ch:char;

procedure count(var a:tlong;
                var dS:integer);
var i:integer;
begin
  ds:=0;
  for i:=1 to n div 2 do
    ds:=ds+a[i]-a[n-i+1];
  end;

procedure out(var a:tlong);
begin
  assign(f,'ticket.out');rewrite(f);
  for i:=1 to n do
    write(f,a[i]);
  close(f);
  halt;
end;

procedure diff(var a,b,c:tlong);
var i:integer;
    t:integer;
begin
  t:=0;
  for i:=n downto 1 do begin
    t:=t+a[i]-b[i]+20;
    c[i]:=t mod 10;
    t:=t div 10-2;
  end;
end;

function cmp(var a,b:tlong):boolean;
var i:integer;
begin
  for i:=1 to n do
    if a[i]<b[i] then begin
      cmp:=a[i]>b[i];
      exit;
    end;
  cmp:=true;
end;

begin
  assign(f,'ticket.in');reset(f);
  n:=0;
  while not eoln(f) do begin
    inc(n);
    read(f,ch);
    x[n]:=ord(ch)-48;
  end;
  count(x,ds);
  if ds=0 then
    out(x);
  {findmore}
  for i:=n downto 1 do begin
    y1:=x;
    if y1[i]=9 then
      continue;
    inc(y1[i]);
    for j:=i+1 to n do
      y1[j]:=0;
    count(y1,ds);
    if (ds>=0)and
      (ds<=9*(n-i)+9-y1[i])
    then begin
      for j:=n downto n-ds div 9+1 do
        y1[j]:=9;
      inc(y1[n-ds div 9],ds mod 9);
      break;
    end;
  end;
  {findless}
  for i:=n downto 1 do begin
    y2:=x;
    if y2[i]=0 then
      continue;
    dec(y2[i]);
    for j:=i+1 to n do
      y2[j]:=9;
    count(y2,ds);
    if (ds<=0)and
      (ds>=-9*(n-i)-y2[i])
    then begin
      ds:=-ds;
      for j:=n downto n-ds div 9+1 do
        y2[j]:=0;
      dec(y2[n-ds div 9],ds mod 9);
      break;
    end;
  end;
  diff(y1,x,d1);
  diff(x,y2,d2);
  if cmp(d1,d2) then
    out(y2)
  else out(y1);
end.

```

## Задача 2. Сломанный калькулятор

Входной файл	calc.in
Выходной файл	calc.out
Ограничение по времени	1 с
Максимальный балл за задачу	100

Ваш младший брат пошёл в первый класс, и вы решили сделать ему скромный подарок — отдать в вечное пользование старый калькулятор. Устройство сделано ещё в СССР, поэтому до текущего дня в рабочем состоянии остались только клавиши  $+$ ,  $-$ ,  $*$ ,  $=$  и некоторые цифры.

Очень скоро младший брат понял, что не все числа можно легко получить на экране калькулятора. Так, если представить, что работают только цифры  $2$  и  $3$ , то число  $55$  можно получить, самое простое, нажав  $2\ 2\ +\ 3\ 3\ =$ .

Для того, чтобы не тратить много времени на ответы брату, как получить то или иное число на экране, вы решили написать программу, которая по заданному числу  $N$  и списку работающих цифровых клавиш находит какую-нибудь (не обязательно кратчайшую) последовательность клавиш, нажатие на которые приводит к  $N$ . Допустимой последовательностью клавиш для калькулятора является любая последовательность клавиш вида  $\langle \text{число} \rangle \langle \text{действие} \rangle \langle \text{число} \rangle \langle \text{действие} \rangle \dots \langle \text{действие} \rangle \langle \text{число} \rangle \langle \text{кнопка равно} \rangle$ . Так, последовательности  $0\ =$  и  $1\ 9\ -\ 0\ 0\ =$  допустимы, а  $*\ 7\ +\ 3\ =$  или  $3\ +\ -\ 2$  — недопустимы.

Напоминаем, что древние калькуляторы не распознают порядок действий, поэтому при нажатии, например,  $2\ +\ 3\ * \ 4\ =$  будет получено  $20$ , а не  $14$ .

### Формат входных данных

В первой строке входного файла указано целое число  $K$  — количество рабочих цифр ( $1 \leq K \leq 10$ ). Во второй строчке перечислены эти самые рабочие цифры, цифры не повторяются. В третьей строке задано целое число  $N$  ( $1 \leq N \leq 1000$ ), которое нужно получить на экране калькулятора.

### Формат выходных данных

Если число  $N$  получить нельзя, то выведите в выходной файл единственное слово 'Impossible', иначе нужно выдать любую допустимую последовательность клавиш, приводящую к требуемому результату. Все

символы в выходном файле должны находиться в первой строке без пробелов. Размер выходного файла не должен превосходить 65536 байт.

### Пример

Входной файл	Выходной файл
3 9 2 3 24	9+3*2=
10 0 1 2 3 4 5 6 7 8 9 1	1234567891234-1234567891233=
1 9 8	Impossible

### Решение

Пусть  $d$  — это наибольший общий делитель работающих цифр в калькуляторе. Понятно, что все числа, составленные из этих цифр, а также их сумма, разность и произведение, будут делиться на  $d$ , поэтому если  $N$  не кратно общему делителю, то сразу говорим 'Impossible'.

Теперь увидим, как в случае делимости  $N$  на  $d$  можно построить ответ.

Можно показать (например, непосредственным перебором), что, если  $\text{НОД}(a_1, a_2, \dots, a_K) = d$ , где  $a_i$  — это цифра, то найдутся такие индексы  $i, j \in \{1, \dots, K\}$  (не обязательно различные) и целые неотрицательные коэффициенты  $p \leq 9$  и  $q \leq 9$ , что  $a_i p - a_j q = d$ , то есть НОД произвольного набора цифр можно представить при помощи суммы и разности максимум двух различных цифр. Заметим, что данный факт в случае произвольных положительных целых  $a_i$  не имеет места.

Теперь можно получить  $d$  на калькуляторе:

$$d = \underbrace{a_i + a_i + \dots + a_i}_p - \underbrace{a_j - a_j - \dots - a_j}_q.$$

После того, как мы научились набирать  $d$ , осталось просуммировать его  $N/d$  раз:

$$N = \underbrace{(a_i + \dots - a_j) + \dots + (a_i + \dots - a_j)}_{N/d \text{ скобок}}.$$

(Сумма в каждой скобке равна  $d$ .)

## Пример правильной программы

```

program calc;
type
  integer = longint;

var
  dig: array[0..9] of boolean;
  _a, _b, _i, _j: integer;
  a, b, i, j: integer;
  min, n: integer;
  s: string;

procedure get(x:integer);
begin
  while x<>0 do begin
    for i:=1 to _a do begin
      write(_i);
      if i<>_a then write('+');
    end;
    for j:=1 to abs(_b) do
      write('-',_j);
    dec(x,min);
    if x<>0 then write('+');
  end;
  writeln('=');
end;

begin
  fillchar(dig,sizeof(dig),false);
  assign(input,'calc.in');
  reset(input);
  assign(output,'calc.out');
  rewrite(output);
  readln(j);
  for i:=1 to j do begin
    read(a);
    dig[a]:=true;
  end;
  min:=MaxLongInt;
  for i:=0 to 9 do if dig[i] then
    for j:=0 to 9 do if dig[j] then
      for a:=1 to 10 do
        for b:=-10 to 0 do
          if (i*a+j*b>0)and
            (i*a+j*b<min) then begin
            min:=i*a+j*b;
            _i:=i; _j:=j;
            _a:=a; _b:=b;
          end;
        read(n);
        if n mod min<>0 then
          writeln('Impossible')
        else get(n);
        close(output);
        close(input);
      end.

```

## Задача 3. Принтер

*Входной файл*

printer.in

*Выходной файл*

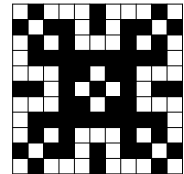
printer.out

*Ограничение по времени*

1 с

*Максимальный балл за задачу*

100



Недавно одна известная фирма решила выпустить чёрно-белый принтер. Для этого принтера решено создать продвинутое программное обеспечение для печати полутоновых (серых) изображений. Такие изображения будут печататься следующим образом. Сначала полутоновое изображение размера  $X \times Y$  пикселей переводится в чёрно-белое изображение размера  $NX \times NY$  (каждый пиксель в полутоновом изображении соответствует некоторому квадрату  $N \times N$  в чёрно-белом изображении; заполнение этого квадрата зависит от цвета пикселя по определённым правилам), потом полученная картинка печатается на принтере.

Правила перевода полутонового пикселя в квадрат  $N \times N$  в чёрно-белом изображении следующие:

- 1) каждая клетка квадрата красится либо в чёрный, либо в белый цвет;
- 2) заполнение квадрата должно быть симметричным относительно обеих диагоналей и относительно двух средних линий квадрата (т. е. линий, соединяющих середины противоположных сторон);
- 3) среди всех таких заполнений нужно выбрать то, в котором отношение числа чёрных клеток  $m$  к площади квадрата находится как можно ближе к яркости полутонового пикселя  $r$  (т. е. модуль разности  $|r - m/N^2|$  должен быть как можно меньше).

Напишите программу, которая по данным  $N$  и  $r$  найдёт необходимое заполнение квадрата  $N \times N$ .

### Формат входных данных

В первой строке входного файла находятся два числа: целое  $N$  и вещественное  $r$  ( $1 \leq N \leq 100, 0 \leq r \leq 1$ ).

### Формат выходных данных

В выходной файл выведите искомое заполнение квадрата  $N \times N$ . А именно, выведите  $N$  строк по  $N$  символов в каждой: символом '.' (точка, код ASCII 46) обозначайте белую клетку, символом '#' («решётка», код ASCII 35) — чёрную.

При наличии нескольких решений выведите любое.

### Пример

Входной файл	Выходной файл	Входной файл	Выходной файл
3 0.4444	<pre> .#.# ... #.#                     </pre>	11 0.5	<pre> .#...#...# #.#.#.#.#.# #.#...#.# ##### ...##.##... ##.#.#.#.## ...##.##... ##### .#.#...#.# #.#.#.#.#.# #...#...#                     </pre>
4 0.498	<pre> .##. #..# #..# .##.                     </pre>		

*Примечание:* Ответ последнего примера соответствует рисунку. Отношение  $m/N^2$  для последнего примера примерно равно 0,50413.

## Решение

Рассмотрим одну восьмую часть квадрата, ограниченную половиной средней линии и половиной диагонали. Назовём её фигурой  $A$ . Поскольку отражениями этой фигуры относительно осей симметрии квадрата можно получить весь квадрат, то её закраска определяет закраску всего квадрата, причём сами клетки фигуры  $A$  можно закрашивать независимо. Не все клетки фигуры  $A$  равноправны — некоторые отвечают за 8 клеток исходного квадрата (назовём их  $A_8$ ), некоторые за 4 ( $A_4$ ), а клетка, находящаяся в центре квадрата (если  $N$  нечётно), при любых отражениях переходит сама в себя ( $A_1$ ). Очевидно, что для любого  $N$  возможно получить яркость  $4k/N^2$ , а для нечётных  $N$  дополнительно можно получить яркость  $(4k+1)/N^2$  (где  $k$  — целое число,  $0 \leq k \leq N^2/4$ ). Выберем из этих чисел ближайшее к  $r$  — тем самым мы определим количество закрашиваемых клеток квадрата  $M$ . После этого начинаем закрашивать клетки, начиная с клеток  $A_8$ , а когда их не хватает, либо когда закраска следующей такой клетки приводит к превышению числа  $M$ , используем клетки  $A_4$  и  $A_1$ . Соответственно, задача сводится к определению того, скольким клеткам исходного квадрата соответствует каждая клетка фигуры  $A$ .

*Примечание:* Отдельно пронумеровав клетки  $A_8$ ,  $A_4$  и  $A_1$ , и определив, сколько клеток какого типа нужно закрасить, можно решить задачу, используя  $O(1)$  памяти. Это решение и приведено в качестве примера.

## Пример правильной программы

```

const inputFile = 'printer.in';
      outputFile = 'printer.out';
type TArray = array[0..2] of longint;
var n,i,j,ii,jj,m,l:longint;
    k,f1,f2,f:TArray;
    a:real;
    av,aa1,aa2:longint;
procedure canon(var i,j:longint);
var t:longint;
begin
  if i>n+1-i then i:=n+1-i;
  if j>n+1-j then j:=n+1-j;
  if i>j then begin
    t:=i; i:=j; j:=t;
  end;
end;
procedure id(i,j:longint;
            var l,m:longint);
var n2:longint;
begin
  if odd(n) then begin
    n2:=n div 2;
    if i=n2+1 then begin
      l:=0; m:=1
    end
    else
      if i=j then begin
        l:=1; m:=i
      end
    else
      if j=n2+1 then begin
        l:=1; m:=n2+i
      end
    else begin
      l:=2; m:=i+((j-1)*(j-2)) div 2;
    end;
  end
  else begin
    if i=j then begin
      l:=1; m:=i
    end
    else begin

```

```

    l:=2; m:=i+((j-1)*(j-2)) div 2;
  end;
end;
end;
begin
  assign(input,inputFile);
  reset(input);
  assign(output,outputFile);
  rewrite(output);
  read(n,a);
  av:=trunc((n*n)*a);
  aa1:=av;
  if odd(n) then begin
    k[0]:=1; k[1]:=n-1;
    k[2]:=((n-3)*(n-1)) div 8;
  end
  else begin
    k[0]:=0; k[1]:=n div 2;
    k[2]:=((n-2)*n) div 8;
  end;
  while (aa1>=8)and
    (f1[2]<k[2]) do begin
    inc(f1[2]); dec(aa1,8);
  end;
  while (aa1>=4)and
    (f1[1]<k[1]) do begin
    inc(f1[1]); dec(aa1,4);
  end;
  while (aa1>=1)and
    (f1[0]<k[0]) do begin
    inc(f1[0]); dec(aa1,1);
  end;
  f2:=f1; aa2:=aa1;
  if f2[0]<k[0] then begin
    inc(f2[0]); dec(aa2,1);
  end
  else if f2[1]<k[1] then begin
    inc(f2[1]); dec(aa2,4);
    inc(aa2,f2[0]); f2[0]:=0;
  end
  else if f2[2]<k[2] then begin
    inc(f2[2]); dec(aa2,8);
    inc(aa2,f2[0]); f2[0]:=0;
    inc(aa2,f2[2]*4); f2[0]:=0;
  end;
  aa1:=av-aa1;
  aa2:=av-aa2;
  if abs(aa2/(n*n)-a)<
    abs(aa1/(n*n)-a) then f:=f2
  else f:=f1;
  for i:=1 to n do begin
    for j:=1 to n do begin
      ii:=i; jj:=j;
      canon(ii,jj);
      id(ii,jj,m,1);
      if l<=f[m] then write('#')
      else write(' ');
    end;
    writeln;
  end;
end.

```

## Задача 4. Сортировка

<i>Входной файл</i>	sort.in
<i>Выходной файл</i>	sort.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Во входном файле находятся  $N$  слов. Отсортируйте их по возрастанию первого символа, сохраняя исходный порядок при совпадении первого символа у слов. Например, слово 'ac' должно идти раньше слов 'be' и 'bb', но не обязательно раньше слова 'ad' или 'a'. В последних двух случаях раньше должно идти то слово, которое находится раньше во входном файле. В выходной файл вы должны вывести только каждое  $M$ -е (т. е.  $M$ -е,  $2M$ -е,  $3M$ -е и т. д.) в отсортированном порядке слово (считая, что после сортировки их занумеровали, начиная с 1).

### Формат входных данных

В первой строке входного файла находятся два целых числа  $N$  и  $M$  ( $1 \leq N \leq 10\,000$ ,  $1 \leq M \leq N$ ). Далее идут  $N$  строк, содержащих слова, которые надо отсортировать. Каждое слово находится на отдельной

строке и имеет длину не более 200 и не менее 1 символа. Слова состоят из маленьких латинских букв.

### Формат выходных данных

В выходной файл выведите требуемые слова по одному на строке. Гарантируется, что в каждом тесте вам придётся вывести не более 200 слов.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
3 2 foo bar ba	ba
3 1 foo bar ba	bar ba foo
676 100 zz zy zx : : ab aa	de hi lm pq tu xy

*Примечание:* В третьем примере во входном файле находятся все  $26^2 = 676$  двухбуквенных слов, отсортированные в порядке, обратном алфавитному.

### Решение

Понятно, что мы не сможем хранить все слова, перечисленные во входном файле, так как в худшем случае объем входной информации составит 10 000 слов по 200 символов, что гораздо больше допустимых размеров памяти. Но всё хранить и не надо. Предложенное ниже решение использует только массив под ответ из максимум 200 строковых переменных длиной не более 200 символов.

Эту задачу можно решить за два прохода по входному файлу. Во время первого прохода для каждой буквы  $\theta$  мы считаем  $L_\theta$  — число слов, начинающихся на эту букву.

Начнём считывать входной файл во второй раз.

Пусть очередное слово  $S$  оказалось  $k$ -м среди слов, начинающихся на букву  $\theta$ , тогда в упорядоченном списке оно займёт место  $k + L_a + L_b + \dots$ , так как впереди него расположены ещё все слова, начинающиеся на буквы, идущие по алфавиту до  $\theta$ . Если  $k + L_a + L_b + \dots = mr$ , то есть позиция кратна  $m$ , то  $S$  нужно вывести в выходном файле на  $r$ -й строчке, поэтому запомним его в  $r$ -м элементе выходного массива `ans`.

После того, как входной файл будет прочитан второй раз, останется лишь вывести массив ответа `ans`.

## Пример правильной программы

```
var f:text;
    s:string;
    ans:array[1..200] of string;
    nn:array[' '..'z'] of integer;
    n,m:integer;
    ch:char;
    i:integer;

begin
  assign(f,'sort.in');reset(f);
  readln(f,n,m);
  fillchar(nn,sizeof(nn),0);
  for i:=1 to n do begin
    readln(f,s);
    inc(nn[s[1]]);
  end;
  close(f);
  nn[' ']:=1;

  for ch:='a' to 'z' do
    nn[ch]:=nn[ch]+nn[pred(ch)];
  for ch:='z' downto 'a' do
    nn[ch]:=nn[pred(ch)];
  reset(f);
  readln(f);
  for i:=1 to n do begin
    readln(f,s);
    if nn[s[1]] mod m=0 then
      ans[nn[s[1]] div m]:=s;
    inc(nn[s[1]]);
  end;
  close(f);
  assign(f,'sort.out');rewrite(f);
  for i:=1 to n div m do
    writeln(f,ans[i]);
  close(f);
end.
```

## Задача 5. Мах

<i>Входной файл</i>	max.in
<i>Выходной файл</i>	max.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Вам дано  $N$  чисел. Выберите из них такие два  $x$  и  $y$ , чтобы их частное  $x/y$  было наибольшим возможным.

### Формат входных данных

В первой строке входного файла находится одно целое число  $N$  — общее количество чисел ( $2 \leq N \leq 30\,000$ ). В следующей строке входного файла находятся  $N$  целых чисел  $a_1, a_2, \dots, a_N$ ; числа не превосходят по модулю 30 000. Гарантируется, что среди этих чисел хотя бы одно не равно нулю.

## Формат выходных данных

В выходной файл выведите сначала число  $x$ , потом  $y$ . Знаменатель  $y$  не должен равняться нулю;  $x$  и  $y$  могут быть одинаковыми, только если соответствующее им число появляется во входном файле более одного раза (строго говоря, должны существовать такие различные индексы  $i$  и  $j$ , что  $1 \leq i, j \leq N$ ,  $x = a_i$  и  $y = a_j$ ).

Если решений несколько, то выведите любое.

## Пример

<i>Входной файл</i>	<i>Выходной файл</i>
5 -2 0 1 2 -1	2 1
3 -1 0 2	0 -1
3 2 2 0	2 2

## Решение

Задача относилась к наименее сложной категории и решалась, например, таким образом: выбираем в качестве знаменателя минимальное из положительных чисел (если есть), числителя — максимальное из оставшихся чисел. Также выбираем вторую пару: в качестве знаменателя максимальное (минимальное по модулю) из отрицательных чисел (если есть), числителя — минимальное из оставшихся чисел. Сравниваем полученные дроби и печатаем в ответ числитель и знаменатель большей из них (или сразу печатаем, если есть только одна из них). Доказательство того факта, что в ответе получается максимальная из возможных дробей, предоставляется читателю в качестве лёгкого упражнения.

## Пример правильной программы

```

{${N+}
const inf='max.in';
      outf='max.out';
var i,n:integer;
    a:array[1..30000]of integer;
    i1,i2,j1,j2:integer;
    s1,s2:comp;

begin
  assign(input,inf);reset(input);
  assign(output,outf);rewrite(output);

```

```

  read(n);
  for i:=1 to n do
    read(a[i]);
  j1:=0; j2:=0;
  for i:=1 to n do begin
    if ((j1=0)or(a[i]<a[j1]))and
      (a[i]>0) then j1:=i;
    if ((j2=0)or(a[i]>a[j2]))and
      (a[i]<0) then j2:=i;
  end;
  i1:=0; i2:=0;

```



## Формат входных данных

В первой строке входного файла находятся два целых числа,  $N$  и  $M$  — размеры залива ( $1 \leq N, M \leq 100$ ). Далее следуют  $N$  строк по  $M$  символов в каждой, описывающих залив. Символы обозначают следующее:

- ‘.’ (точка, код ASCII 46) — клетка моря;
- ‘#’ («решётка», код ASCII 35) — клетка острова;
- ‘&’ (амперсанд, код ASCII 38) — клетка моря, где происходит старт;
- ‘Z’ (латинская заглавная буква Z, код ASCII 90) — клетка острова,

в которой установлен шест.

Во входном файле будет ровно один символ ‘&’ и ровно один символ ‘Z’.

## Формат выходных данных

Если обогнуть шест возможно, то выведите в выходной файл последовательность символов ‘U’, ‘L’, ‘R’ и ‘D’, описывающую искомый маршрут. Символы обозначают следующее:

- ‘U’ — перемещение на клетку вверх;
- ‘L’ — влево;
- ‘R’ — вправо;
- ‘D’ — вниз.

Среди всех решений необходимо выбрать решение, содержащее как можно меньше символов. Если существует несколько оптимальных решений, можно вывести любое из них.

Если обогнуть шест невозможно, выведите в выходной файл единственную строку ‘Impossible’ (без кавычек).

**Пример**

<i>Входной файл</i>	<i>Выходной файл</i>
4 5 ..... &Z#.. .#... ...##	URRRDDL DLLUU
5 5 ..... .&##. .#Z.. ..#.. .....	URRRDDDD LLLL UUU
5 4 &#.. #... ..Z. .##. ....	Impossible

*Примечание:* Второй пример соответствует рисунку.

**Решение**

Возьмём две одинаковые карты моря и проведём на обеих картах из точки, в которой находится флаг, горизонтальный луч вправо (вообще, направление луча значения не имеет). После этого разрежем карты по нарисованным лучам и склеим крест-накрест так, чтобы при пересечении луча мы переходили с первой карты на вторую и наоборот (к сожалению, в трёхмерном пространстве, в котором мы вынуждены находиться, такая склейка невозможна<sup>1</sup>, но идея вам должна быть ясна). Все пути из точки старта на первом листе в точку старта на втором обходят вокруг флага нечётное число раз, а все пути, начинающиеся и оканчивающиеся в точке старта на одном из листов,— чётное. Поскольку кратчайшим среди путей, обходящих флаг, является путь с одним оборотом (это несложно доказать), то для получения ответа найдём поиском в ширину кратчайший путь с точки старта на первой карте в точку старта на второй.

<sup>1</sup> На самом деле, в трёхмерном пространстве *можно* произвести некоторый эквивалент такой склейки, но мы его описывать здесь не будем.

## Пример правильной программы

```

{$M 65520,0,1024}
const
di:array[1..4] of integer=(-1,1,0,0);
dj:array[1..4] of integer=(0,0,-1,1);
pr:array[1..4] of integer=(2,1,4,3);
let:array[1..4] of char='DURL';
MaxN=100;
MaxLen=MaxN*MaxN*3;
var a:array[0..MaxN+1,0..MaxN+1,0..1]
      of integer;
      from:array[1..MaxN,1..MaxN,0..1]
            of byte;
f:text;
si,sj,i,j,k,i0,j0,n,m:integer;
ch:char;

procedure move(i,j,k,l:integer;
              var ii,jj,kk:integer);
begin
  ii:=i+di[l];jj:=j+dj[l];
  kk:=k;
  if (i=i0)and(j<j0)and(l=1) then
    kk:=1-kk;
  if (i=i0-1)and(j<j0)and(l=2) then
    kk:=1-kk;
end;

procedure search;
var q:array[1..MaxN*MaxN*2] of record
      i,j,k:byte;
    end;
  i,j,k,ii,jj,kk,b:integer;
  cl:integer;
  l,r:integer;
begin
  q[1].i:=si;q[1].j:=sj;q[1].k:=0;
  l:=1;r:=1;
  while l<r do begin
    i:=q[l].i;
    j:=q[l].j;
    k:=q[l].k;
    cl:=a[i,j,k];
    inc(l);
    for b:=1 to 4 do begin
      move(i,j,k,b,ii,jj,kk);
      if a[ii,jj,kk]>cl+1 then begin
        a[ii,jj,kk]:=cl+1;
        from[ii,jj,kk]:=b;
        inc(r);
        q[r].i:=ii;
        q[r].j:=jj;
        q[r].k:=kk;
      end;
    end;
  end;
begin
  assign(f,'sea.in');reset(f);
  readln(f,n,m);
  fillchar(a,sizeof(a),255);
  for i:=1 to n do begin
    for j:=1 to m do begin
      read(f,ch);
      case ch of
        '.': begin
          a[i,j,0]:=MaxLen;
          a[i,j,1]:=MaxLen;
        end;
        '&': begin
          a[i,j,0]:=0;a[i,j,1]:=MaxLen;
          from[i,j,0]:=0;
          si:=i; sj:=j;
        end;
        'Z': begin
          i0:=i; j0:=j;
        end;
      end;
    end;
  end;
  readln(f);
  search;
  assign(f,'sea.out');rewrite(f);
  if a[si,sj,1]<MaxLen then begin
    i:=si; j:=sj; k:=1;
    repeat
      if from[i,j,k]=0 then break;
      write(f,let[from[i,j,k]]);
      move(i,j,k,pr[from[i,j,k]],i,j,k);
    until false;
    end else writeln(f,'Impossible');
  close(f);
end.

```

## Результаты II Нижегородской городской олимпиады школьников по информатике

	Школа	Кл.	1	2	3	4	5	6	= Дипл.
1. P77: Разенштейн Илья	лиц. 40	10	63	100	100	100	100	100	563 I
2. P72: Мишенин Александр	лиц. 40	11	62	100	100	68	100	28	458 I
3. P16: Калинина Елена	лиц. 3, Саров	11	63	92	100	61	100		416 I
4. P79: Епифанов Владислав	лиц. 40	8	56	96	88	69	70	34	413 I
5. P73: Муравьев Александр	лиц. 40	11	54	69	100	69	98	0	390 I
6. P30: Голубев Кирилл	165	11	43	63	72	76	100	0	354 I
7. P18: Семенов Станислав	лиц. 15, Саров	9	56	74	100	10	100	0	340 I
8. P19: Побуринная Оксана	гимн. 2, Саров	9	49		100	84	96		329 I
8. P71: Бударагин Дмитрий	лиц. 40	11	0	93	76	76	84	0	329 I
10. P40: Мхитарян Сергей	63	11	18	100	88	76	24	0	306 II
11. P36: Багиев Александр	165	11	29	77	76	41	80	0	303 II
12. P35: Песков Александр	165	11	0		94	92	100		286 II
13. P38: Тураев Марат	36	10	13	51	4	84	98	0	250 II
14. P39: Матросов Михаил	63	11	39	0	68	54	88	0	249 II
15. P76: Рамин Виктор	лиц. 40	11	18	0	54	76	100	0	248 II
16. P31: Низовцев Сергей	165	8		93		100	40	0	233 II
17. P17: Урбанович Аркадий	лиц. 3, Саров	11	46	16		69	100		231 II
18. P45: Хусайнов Тимур	85	10	56			92	82		230 II
19. P33: Погорелов Дмитрий	165	8		58		69	100	0	227 II
20. P34: Самойлыч Максим	165	10	56		78	76	10		220 II
21. P67: Шумилов Вячеслав	лиц. 38	11	52			68	98		218 II
22. P22: Макарец Сергей	лиц. 15, Саров	11	56			61	100		217 II
23. P07: Крылова Ирина	лиц. 87	8	49			29	96	14	188 III
24. P23: Борискин Павел	лиц. 3, Саров	10	49	48		68	12		177 III
25. P58: Соколов Андрей	173	11	43	30			100	0	173 III
26. P42: Смирнов Антон	82	10	16	51		6	96	0	169 III
27. P09: Шеллунц Артур	180	10	56			92	18		166 III
28. P37: Брандт Андрей	165	10	56	0	4		98	0	158 III
29. P84: Бастраков Сергей	67	11	8	11	4	27	96	0	146 III
30. P56: Федоров Сергей	лиц. 38	11	29	15	0	3	98		145 III
31. P44: Амосов Дмитрий	82	11	49	0			90		139 III
32. P41: Сопин Дмитрий	82	11	49	0	12	16	60	0	137 III

33.	P32: Тихонов Андрей	165	8	43	0	8	61	24	0	136	III
34.	P55: Скляров Олег	лиц. 38	11	24	7	22	68	12	0	133	III
35.	P74: Парахоняк Анастасия	лиц. 40	11	17			10	100	0	127	III
36.	P78: Булатов Алексей	лиц. 40	9	56				64		120	III
37.	P64: Дружков Павел	17	11	29	0	0	76	10	0	115	III
38.	P53: Збруев Дмитрий	лиц. 38	11	0	0	4	84	18	0	106	III
39.	P52: Бронников Вячеслав	лиц. 38	11	56	19	16	7	6		104	III
40.	P20: Огнева Дарья	лиц. 15, Саров	8	10			68	24		102	III
41.	P93: Февральских Сергей	121	10	49			37	12		98	
42.	P15: Машуков Сергей	94	11	56	28			10		94	
43.	P13: Шевченко Андрей	91	11	40			36	12		88	
44.	P82: Банин Максим	117	10		0		84	0	0	84	
45.	P61: Чистяков Андрей	32	8	43	0		16	24	0	83	
46.	P75: Путилов Алексей	лиц. 40	11	11	0	26	3	34	0	74	
47.	P06: Бобылев Дмитрий	лиц. 87	10	56			6	10	0	72	
48.	P43: Катичев Роман	82	11	43		4	9	10	0	66	
49.	P04: Бабаев Дмитрий	лиц. 87	11	49			0	12	0	61	
50.	P92: Савинов Сергей	121	10	12	8		0	34		54	
51.	P91: Дегтярев Владимир	121	10	7			40	6		53	
52.	P94: Кузнецов Александр	110	11	25	0	0	11	12		48	
53.	P10: Маслеников Николай	180	11	29	3			10		42	
53.	P59: Татур Сергей	СОШ 24	11	23	4		3	12		42	
53.	P62: Щебентовский Андрей	32	11	30	0			12	0	42	
56.	P95: Ершов Андрей	лиц. 40	10	0	0	18	0	22	0	40	
57.	P01: Каратаев Денис	гимн. 80	11	14			9	12		35	
57.	P63: Моисеев Александр	17	11	15			6	14		35	
59.	P05: Насеткин Иван	лиц. 87	11	15	0	0	10	6		31	
60.	P51: Бовыкин Максим	лиц. 38	11	8	0			18	0	26	
61.	P54: Клишин Алексей	лиц. 38	11	13	0		11	0		24	
62.	P12: Шабанов Дмитрий	180	11	17	0		0			17	
62.	P21: Анисимова Светлана	гимн. 2, Саров	7	17				0		17	
64.	P57: Борясяк Алексей	лиц. 28	11	0	0		6	4		10	
64.	P65: Шельцин Арсений	154	10	0				10		10	

В таблице не указаны участники, набравшие менее 10 баллов (21 человек).



# III городская олимпиада школьников по информатике 30 января 2007 г.

## Авторы задач

Бударагин Д. И., студент 1 курса ВШОПФ ННГУ;  
Калинин П. А., студент 5 курса ВШОПФ ННГУ;  
Круглов А. А., аспирант ИПФ РАН;  
Лелюх В. Д., старший преподаватель ННГУ;  
Тимушев Р. И., студент 4 курса ВШОПФ ННГУ;  
Хаймович И. М., студент 4 курса ВШОПФ ННГУ.

## Задача 1. Бассейны и трубы

<i>Входной файл</i>	pools.in
<i>Выходной файл</i>	pools.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

В новом аквапарке есть  $N$  бассейнов, заполненных водой. Некоторые пары бассейнов соединены трубами, всего труб  $M$ . Система труб построена так, чтобы из любого бассейна в любой другой можно было при необходимости перегнать воду, возможно, воспользовавшись другими бассейнами как промежуточными. На каждой трубе стоит программируемый насос, который способен перекачивать в любую сторону произвольный заданный поток воды. Кроме того, в некоторые бассейны дополнительно втекает вода с известной скоростью, а из некоторых вода вытекает — тоже с известной скоростью.

Ваша задача — написать программу, которая отрегулирует насосы так, чтобы уровень воды в бассейнах поддерживался постоянным.

### Формат входных данных

В первой строке указано количество бассейнов  $N$  ( $1 \leq N \leq 100$ ). Далее следуют  $N$  строк, содержащих информацию о бассейнах. Для каждого бассейна, начиная с 1-го и кончая  $N$ -м, указана разница между дополнительным втекающим и вытекающим потоком — целое число, не

превышающее по модулю 30 000 (положительное число соответствует втеканию, отрицательное — вытеканию).

На следующей строке входного файла находится число труб  $M$  ( $0 \leq M \leq 4950$ ). В последующих  $M$  строках находится описание труб: для каждой трубы указаны два числа — номера бассейнов, которые труба соединяет. Между каждой парой бассейнов есть не более одной трубы, трубы не могут соединять бассейн с самим собой.

### Формат выходных данных

Если решение существует, выведите в выходной файл  $M$  строк, по числу на строку. Для каждой трубы в том же порядке, как во входном файле, должно быть указано целое число — поток воды через неё. Положительным числом обозначается поток от бассейна, указанного первым во входном файле, ко второму; отрицательным — обратный поток. Поток в выходном файле не должны превышать 2 000 000 000.

Если возможно несколько решений, можно выводить любое из них.

В случае, когда решения не существует, выходной файл должен содержать одну строку 'Impossible'.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
4 -100 -100 100 200 4 1 2 4 3 1 3 4 1	Impossible
4 -100 -100 100 100 4 1 2 4 3 1 3 4 1	100 0 -100 100

## Решение

У задачи в такой постановке может быть большое число правильных ответов, что может создавать сложности с придумыванием принципа выбора из них *одного* ответа, как этого требуют условия.

Естественно построить неориентированный граф, вершинами которого являются бассейны, а рёбрами — трубы. Неоднозначность ответа возникает из-за возможного наличия в этом графе циклов: вдоль каждого цикла насосы могут перекачивать произвольное количество воды, не нарушая правильности ответа.

Одна из идей, позволяющих выбрать из ответов один, следующая: выкинуть лишние рёбра в графе так, чтобы избавиться от циклов, но сохранить связность графа (т. е. сохранить условие, что из любого бассейна в любой другой можно перегнать воду). Такой подграф называют *остовным деревом* данного графа. Остовное дерево можно строить, запустив по графу *поиск в глубину*. Поиск в глубину — это рекурсивный обход всех вершин графа: при обработке очередной вершины мы вызываем процедуру обхода для каждой ещё не обработанной вершины, связанной ребром с текущей. Рёбра, по которым мы переходим при таком построении, образуют остовное дерево.

Каждое ребро дерева разделяет его на две части (иначе дерево содержало бы цикл), поэтому поток воды, текущий по этому ребру, должен быть равен избытку жидкости в одной из частей. Этот избыток удобно подсчитывать прямо в процедуре, осуществляющей поиск в глубину. Легко проверить, что таким образом отрегулированные насосы уравнивают избыток (или недостаток) воды в *каждом* из бассейнов в случае, когда суммарный избыток воды (или недостаток с обратным знаком) во всех бассейнах равен нулю. В случае, когда он отличен от нуля, очевидно, ответ 'Impossible'.

## Пример правильной программы

```

const maxn=100;
var n,m:integer;
    xs:array[1..maxn]of integer;
    flow:array[1..maxn,1..maxn]of longint;
    es:array[1..maxn]of set of 1..maxn;
    was:array[1..maxn]of boolean;
    totx:longint;

procedure readin;
var i:integer;
    a,b:integer;
begin
    fillchar(es,sizeof(es),0);
    readln(n);
    for i:=1 to n do
        readln(xs[i]);
    readln(m);
    for i:=1 to m do begin
        readln(a,b);
        include(es[a],b);
        include(es[b],a);
    end;
end;

function deepfirstsearch(i:integer):longint;
var j:integer;

```

```

    sumx,childx:longint;
begin
    was[i]:=true;
    sumx:=xs[i];
    for j:=1 to n do
        if (j in es[i])and not was[j] then begin
            childx:=deepfirstsearch(j);
            flow[j,i]:=childx;
            flow[i,j]:=-childx;
            inc(sumx,childx);
        end;
    deepfirstsearch:=sumx;
end;

procedure solve;
begin
    fillchar(was,sizeof(was),0);
    fillchar(flow,sizeof(flow),0);
    totx:=deepfirstsearch(1);
end;

procedure writeout;
var i:integer;
    a,b:integer;
begin
    if totx<>0 then begin
        writeln('Impossible');
    end
    else begin
        reset(input);
        for i:=1 to n+2 do readln;
        for i:=1 to m do begin
            readln(a,b);
            writeln(flow[a,b]);
        end;
    end;
end;

begin
    assign(input,'pools.in');
    reset(input);
    readln;
    solve;
    assign(output,'pools.out');
    rewrite(output);
    writeout;
end.
```

## Задача 2. Батон

<i>Входной файл</i>	baton.in
<i>Выходной файл</i>	baton.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Несколько школьников решили отпраздновать Новый Год вместе. Для празднования они закупили гору продуктов, и в том числе батон колбасы. Количество еды было настолько велико, что об этом батоне вспомнили только через два с лишним часа после боя курантов. И, конечно, в честь столь знаменательного момента времени школьники решили как можно быстрее съесть этот батон. Но на этом пути их поджидала одна трудность: батон надо разрезать на  $N$  (по числу школьников) равных частей, причём как можно быстрее, чтобы не упустить момент. Для этой цели нашёлся очень длинный и острый нож, который смог бы при необходимости разрезать даже  $N$  батонов колбасы за один раз. Кроме того, нашлась линейка, с помощью которой можно отмерять любую наперёд заданную часть батона, выбирая место будущего разреза с достаточной точностью. Батон колбасы оказался настолько тонким и длинным, что резать его имело смысл только перпендикулярно его оси.

Конечно, школьники быстро сообразили, что для ускорения процесса можно резать несколько кусков батона одновременно. Батон колбасы оказался прямым и негнувшимся, поэтому сэкономить число действий за счёт складывания батона перед разрезанием нельзя. Им осталось определить минимальное количество движений ножом, с помощью которых можно осуществить их намерения. Помогите им в этом.

В процессе разрезания школьник может приложить некоторые имеющиеся у него куски батона колбасы друг к другу как он хочет (напомним, что батон очень длинный и тонкий, поэтому имеет смысл прикладывать куски только параллельно друг другу), и одним движением ножа разрезать все куски. Изначально у школьника есть один кусок — сам батон; в конце у него должно быть  $N$  одинаковых по длине кусков (каждый длиной в  $1/N$  начального батона).

Например, разрезать батон на 6 частей ( $N = 6$ ) школьник может всего за 3 действия ножом:

1. Отмерить  $1/3$  длины всего батона и в этом месте разрезать батон.
2. Большой из получившихся кусков разрезать пополам (в итоге получаются три куска равной длины).
3. Приложив их вплотную друг к другу, выровнять концы и разрезать все три куска посередине, в результате чего получится шесть одинаковых кусков.

Чтобы разрезать батон на 5 равных частей ( $N = 5$ ), также необходимо 3 действия ножом, например:

1. Отмеряем  $2/5$  длины батона и разрезаем батон в этом месте.
2. Получившиеся куски складываем вместе так, чтобы середина меньшего куска находилась против трети большего, и разрезаем их так, чтобы меньший разрезать пополам, а больший — в отношении  $1 : 2$ . Получаем три куска длиной в  $1/5$  всего батона и один — длиной  $2/5$ .
3. Большой из имеющихся кусков разрезаем пополам и получаем пять одинаковых кусков.

## Формат входных данных

Во входном файле находится одно число  $N$  ( $1 \leq N \leq 2\,000\,000\,000$ ).

## Формат выходных данных

Выведите в выходной файл одно число — минимальное количество движений ножом.

## Пример

Входной файл	Выходной файл
6	3
5	3

## Решение

Определим, для каких  $N$  решением является число, не превосходящее  $k$ , т. е. на сколько частей можно разрезать батон, затратив не более  $k$  движений. Во-первых, очевидно, что, т. к. каждый разрез не более чем удваивает число кусков, то за  $k$  разрезов мы получим не более  $2^k$  кусков. Поэтому должно быть как минимум  $N \leq 2^k$ .

Докажем, что, если  $N \leq 2^k$ , то за  $k$  движений *можно* разрезать батон на  $N$  частей. Если  $N = 2^k$ , то всё очевидно: режем батон пополам, потом за одно движение обе половины режем ещё пополам и т. д. Если же  $N < 2^k$ , то приставим к нему фиктивный кусок длины  $(2^k - N)/N$  от начального батона (т. е. сделаем так, чтобы новый батон надо было разрезать на  $2^k$  кусков, получив куски той же длины, что и в начальной задаче). Теперь алгоритм разрезания нового батона на  $2^k$  кусков за  $k$  движений легко преобразовать в алгоритм разрезания начального батона на  $N$  кусков.

Таким образом, осталось определить минимальное  $k$ , для которого  $2^k \geq N$ . Это сделать несложно, например, возводя двойку в степень, но при таком способе решения возникает проблема переполнения 32-битного целого. Альтернативным способом решения является последовательное *деление*  $N$  на два; при этом надо только аккуратно осознать, *что* происходит, чтобы правильно обрабатывать случаи, когда  $N = 2^k$ . Именно этот способ и реализован в примере программы.

## Пример правильной программы

```

var n:longint;
    r:integer;
begin
  assign(input,'baton.in'); reset(input);
  readln(n);
  assign(output,'baton.out');
  rewrite(output);
  r:=0;
  while n>1 do begin
    { inv: k(init N) = r + k(current N) }
    inc(r);
    n:=(n+1)div 2;
  end;
  writeln(r);
  close(output);close(input);
end.
```

## Задача 3. Обфускатор

<i>Входной файл</i>	obfuscat.in
<i>Выходной файл</i>	obfuscat.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Жюри Нижегородской городской олимпиады недавно придумало новый язык программирования, условно названный NN# и предназначенный для выполнения простых операций со строками и числами.

Напишите обфускатор для этого языка. А именно, ваша программа должна сделать следующее:

- заменить все символы перевода строки и комментарии на пробелы;
- после этого удалить все незначащие пробелы и
- переименовать все переменные следующим образом: первую встретившуюся в программе переменную нужно переименовать в **a**, вторую — в **b** и т. д., 26-ю — в **z**, 27-ю — в **aa**, далее **ab**, ..., **az**, **ba**, **bb**, ..., **zz**, **aaa**, **aab**, ...

Если ваша программа будет выполнять только часть этих требований (не меняя при этом смысла программы), она всё равно получит не меньше 50 баллов.

**Синтаксис языка NN#** следующий:

1. Язык является нечувствительным к регистру, т. е. заглавные и маленькие буквы не различаются.
2. Программа начинается со слова **program**, после которого идёт БЛОК-КОДА.
3. БЛОК-КОДА — это последовательность КОМАНД, заключённая в фигурные скобки ('{' и '}').
4. КОМАНДА — это: ОПЕРАТОР-ПРИСВАИВАНИЯ, либо ОПЕРАТОР-ВВОДА, либо ОПЕРАТОР-ВЫВОДА, либо УСЛОВНЫЙ-ОПЕРАТОР. После каждой КОМАНДЫ, кроме УСЛОВНОГО-ОПЕРАТОРА, должен следовать символ ';'.
5. ОПЕРАТОР-ПРИСВАИВАНИЯ записывается следующим образом: ИМЯ-ПЕРЕМЕННОЙ, символ '=', ВЫРАЖЕНИЕ.
6. ОПЕРАТОР-ВВОДА — это слово **read**, после которого идёт ИМЯ-ПЕРЕМЕННОЙ.
7. ОПЕРАТОР-ВЫВОДА — это слово **print**, после которого идёт ВЫРАЖЕНИЕ.
8. УСЛОВНЫЙ-ОПЕРАТОР — это слово **if**, потом — заключённое в круглые скобки условие (ИМЯ-ПЕРЕМЕННОЙ, один из символов '<', '>', '=', потом ИМЯ-ЕЩЁ-ОДНОЙ-ПЕРЕМЕННОЙ), потом — БЛОК-КОДА.

9. **ВЫРАЖЕНИЕ** — это произвольное арифметическое выражение, содержащее круглые скобки, операторы '+', '-', '\*' и '/', ИМЕНА-ПЕРЕМЕННЫХ, СТРОКОВЫЕ- и ЧИСЛОВЫЕ-КОНСТАНТЫ, а также ВЫЗОВЫ-ФУНКЦИЙ.

10. **ЧИСЛОВАЯ-КОНСТАНТА** — это произвольная последовательность десятичных цифр, перед которой может идти символ '+' либо '-'.

11. **СТРОКОВАЯ-КОНСТАНТА** — это заключённая в кавычки ("") последовательность символов. В ней допускаются любые символы, кроме '\ ' и '"', которые также могут встречаться, но только в виде последовательности '\ '. Например, "some text", "\"\" и "text\"more text" — это допустимые СТРОКОВЫЕ-КОНСТАНТЫ, а "\" и "text \\text" — нет.

12. **ВЫЗОВ-ФУНКЦИИ** — это ИМЯ-ФУНКЦИИ, потом заключённый в круглые скобки список аргументов — последовательность ВЫРАЖЕНИЙ, разделённых запятыми (',').

13. **ИМЯ-ФУНКЦИИ**, равно как и **ИМЯ-ПЕРЕМЕННОЙ** — это произвольная последовательность латинских букв.

14. **ПРОБЕЛЬНАЯ-ПОСЛЕДОВАТЕЛЬНОСТЬ** — это произвольная последовательность пробелов, переводов строк и КОММЕНТАРИЕВ. Она *может* появляться в любом месте программы, кроме как внутри ИМЁН-ПЕРЕМЕННЫХ и слов типа `print`, и *должна* быть: между словом `read` и ИМЕНЕМ-ПЕРЕМЕННОЙ, а также между словом `print` и ВЫРАЖЕНИЕМ, если последнее начинается с букв.

15. Переводы строк в программе допускаются *только* в виде комбинации CR+LF, т. е. в виде последовательности из двух символов — с ASCII-кодами 13 и 10 (стандартный перевод строки в DOS и Windows). Эти символы не могут появляться в программе в других комбинациях.

16. В программе допустимы два типа КОММЕНТАРИЕВ: СТРОЧНЫЙ и длинный.

17. **СТРОЧНЫЙ-КОММЕНТАРИЙ** начинается с последовательности символов '/' и продолжается до конца текущей строки.

18. **ДЛИННЫЙ-КОММЕНТАРИЙ** начинается последовательностью символов '/\*' и заканчивается последовательностью символов '\*/'. Начало и конец этого комментария не могут перекрываться, т. е. '\*/' — это *не* комментарий (а начало комментария). Длинными-комментариями являются, например, `/*this is some comment*/` и `/**/`.

19. Начало комментария внутри другого комментария игнорируется, т. е., например, `/*/**/` — это цельный длинный-комментарий, а `///*/` — начало СТРОКОВОГО-КОММЕНТАРИЯ.

20. *Значащими* считаются пробелы внутри СТРОКОВЫХ-КОНСТАНТ,

а также после слова `read` в ОПЕРАТОРЕ-ВВОДА и после слова `print` в ОПЕРАТОРЕ-ВЫВОДА, если ВЫРАЖЕНИЕ в нём начинается с буквы.

### Формат входных данных

Во входном файле находится синтаксически корректная программа на языке NN#. Размер программы не превышает 32 Кб. Число различных переменных в программе не превышает 1000, длина имени каждой не больше 32 символов. Все символы во входном файле имеют ASCII-коды от 32 до 127 включительно (кроме переводов строк, которые, как уже было сказано, задаются последовательностью CR+LF).

### Формат выходных данных

Выведите в выходной файл программу, полученную из исходной в результате необходимых преобразований. Допускается изменение регистра символов, если оно не влияет на работу программы.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
<pre> ProGram_{ rEad/**/X; print_{if+20*( "This/**/_is_some\"\"texT\"\" _{+(-10)); print x; if_(a&lt;b){ _{if_(B&lt;a)_{/**/test _{print_ "Wow!_That's_a_Wonder"; }_{ }/*_this_is a_comment*/ a=b+c+d+a("text"/10,10+10) _{-20*"_"_ "; print_(a+x); } </pre>	<pre> proGraM{rEad_{a;print_{B+20*( "This/**/_is_some\"\"texT\"\" +(-10));print_{a;if(c&lt;d){ if(d&lt;c){print "Wow!_{That's_{a}_{Wonder"}}} c=d+e+f+a("text"/10,10+10) -20*"_"_ ";print(c+a);} </pre>

*Примечание:* Символ ‘\_’ обозначает пробел.

*Примечание:* Для удобства чтения пример выходного файла разбит на строки. На самом деле выходной файл в примере должен состоять из

двух строк с единственным разрывом строки — там, где в примере находится символ ¶ (сам этот символ, конечно, не должен присутствовать в выходном файле).

## Решение

Эта задача носила скорее технический характер, её полное решение могло потребовать значительных усилий. В то же время, несложно было написать решение, проводящее не все требуемые преобразования — по условию, такие программы тоже набирали некоторое количество баллов.

Наиболее просто задачу можно было решить, если обратить внимание на следующие факты:

- если после последовательности букв стоит открывающая скобка, то это не имя переменной;
- последовательность букв, слева от которой (возможно, отделённый пробелом) стоит символ '{' или ';', а справа — не символ '=', тоже не является именем переменной;
- все остальные последовательности букв, не находящиеся внутри строк, — имена переменных.

Вооружившись этим знанием, а так же классическим методом решения задач на синтаксический разбор<sup>1</sup>, достаточно просто написать правильное решение; такое решение и приведено в качестве примера.

Следует заметить, что имена переменных приходится хранить в хеш-таблице, так как в противном случае решение не укладывается в ограничение по времени.

Возможен и другой подход к решению задачи, в данном случае приводящий к несколько более длинному решению, однако довольно полезный на практике, а главное — интуитивно понятный. Условно его можно назвать методом *потока данных*. Решение, использующее этот

---

<sup>1</sup> Во входном файле выделяется дополнительный «смысловый» уровень, на котором некоторые наборы символов объединяются в *лексемы (токены)*, а некоторые — пропускаются. В нашем случае лексемами будут имена переменных и ключевые слова, состоящие из букв, и одиночные символы, не являющиеся латинскими буквами. При грамматическом разборе программы вместо чтения символов напрямую из файла вызывается процедура, считывающая следующую лексему из входного файла (процедура `readtok` в примере программы). Это позволит при разборе входных данных уже не учитывать пробелы и комментарии, т. к. их можно не считать лексемами и пропускать в процедуре чтения. При этом следует особо обрабатывать строки. В предложенном решении все символы, содержащиеся внутри строки, считаются отдельным типом лексем, причём пробелы и комментарии внутри строк, естественно, пропускаются не должны.

подход (tir\_g.pas), можно скачать в числе решений жюри с сайта <http://olympiads.nnov.ru>.

## Пример правильной программы

```

type
tTokTyp=(_readprint,_var,_other,_eof,_eop);
const MAXHASH = $1FFF;
var f,g:text;
    varname:array[1..1000] of string[32];
    hash:array[0..MAXHASH] of integer;
    nvar:integer;
    prevtoktyp,toktyp:tTokTyp;
    tok:string;
    subs:array[-1..1] of char;
    InLongString:boolean;

function lower(ch:char):char;
begin
if ch in ['A'..'Z'] then
    lower:=chr(ord(ch) or 32)
else lower:=ch;
end;

procedure advance;
begin
subs[-1]:=subs[0];
subs[0]:=subs[1];
if eof(g) then
    subs[1]:=#26
else read(g,subs[1]);
end;

procedure readcomment;
begin
advance;advance;
repeat
    advance;
until (subs[0]='/')and(subs[-1]='*');
advance;
end;

procedure readcomment2;
begin
repeat
    advance;
until (subs[0] in [#13,#10,#26]);
advance;
end;

procedure skip;
begin
repeat
if subs[0] in [' ',#13,#10] then advance
else if (subs[0]='/')and(subs[1]='*') then
    readcomment
else if (subs[0]='/')and(subs[1]='/') then
        readcomment2
        else break;
until false;
end;

procedure readstring;
begin
toktyp=_other;
InLongString:=true;
tok:=subs[0];
advance;
end;

procedure ReadLSpert;
begin
if (subs[0]='"')and(subs[-1]<>'\'') then begin
    tok:=subs[0];
    advance;
    InLongString:=false;
end else begin
    tok:=subs[0];
    advance;
end;
end;

procedure readword;
begin
toktyp=_var;
if subs[0] in ['a'..'z','A'..'Z'] then
    while subs[0] in ['a'..'z','A'..'Z'] do
        begin
            tok:=tok+lower(subs[0]);
            advance;
        end
    else begin
        tok:=lower(subs[0]);
        toktyp=_other;
        advance;
    end;
end;

procedure readtok;
begin
if InLongString then begin
    ReadLSpert;
    exit;
end;
tok:='';
skip;
if subs[0]='"' then begin
    readstring;
    exit;

```

```

end else readword;
skip;
if subs[0]='(' then
  toktyp:=_other
else if (subs[0]<>') and ((tok='read') or
  (tok='print')) and (prevtoktyp=_eop) then
  toktyp:=_readprint;
if tok=#26 then
  toktyp:=_eof;
if (tok=';' or (tok='{') or (tok=}')') then
  toktyp:=_eop;
end;

function formvarname(i:integer):string;
var res:string;
begin
if (1<=i) and (i<=26) then
  res:=chr(96+i)
else if (i>=26) and (i<=26*26+26) then begin
  dec(i,27);
  res:=chr(97+i div 26)+chr(97+i mod 26);
end else begin
  dec(i,26+26*26+1);
  res:=chr(97+i div 26 div 26)+
  chr(97+i div 26 mod 26)+
  chr(97+i mod 26);
end;
formvarname:=res;
end;

function hashvar(s:string):integer;
var i:integer;
h:longint;
begin
h := 0;
for i:=1 to length(s) do
  h:=(h*37+ORD(s[i])*17) and MAXHASH;
hashvar:=h;
end;

function findvar(s:string):integer;
var h:integer;
begin
h := hashvar(s);
while hash[h]<>0 do begin
  if (varname[hash[h]]=s) then break;
  inc(h);
  if (h>MAXHASH) then h:=0;
end;
findvar:=h;
end;

function getvarname(s:string):string;
var i,n:integer;
begin
for i:=1 to length(s) do s[i]:=lower(s[i]);
n:=findvar(s);
if (hash[n]=0) then begin
  inc(nvar);
  varname[nvar]:=s;
  hash[n]:=nvar;
end;
getvarname:=formvarname(hash[n]);
end;

begin
assign(g,'obfuscat.in');reset(g);
fillchar(subs,sizeof(subs),0);
advance;
advance;
assign(f,'obfuscat.out');rewrite(f);
nvar:=0;
skip;
readword;
write(f,'program');
prevtoktyp:=_other;
InLongString:=false;
fillchar(hash,sizeof(hash),0);
while true do begin
  readtok;
  if (prevtoktyp=_readprint) and
    (tok[1] in ['a'..'z']) then
    write(f,' ');
  case toktyp of
    _readprint:write(f,tok);
    _var:write(f,getvarname(tok));
    _other,_eop:write(f,tok);
    _eof:break;
  end;
  prevtoktyp:=toktyp;
end;
close(f);
end.

```

## Задача 4. А ты купи коня!

<i>Входной файл</i>	horses.in
<i>Выходной файл</i>	horses.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Вашего друга, который до недавних пор работал кондуктором в маршрутке, повысили, и с этих самых недавних пор он является главным

экономистом в фирме, владеющей этими маршрутками. К сожалению, он очень быстро обнаружил, что его знаний недостаточно для такой работы, и стал срочно навёрстывать упущенное. К счастью, это оказалось не столь сложно, благо изучать ему требовалось не всю экономику, а только ту её часть, которая имеет отношение к рынку маршруток в крупных городах.

Среди множества моделей, предложенных экономистами для описания различных видов рынков, вашему другу больше всего понравилась знаменитая «модель конного рынка» Е. фон Бём-Баверка. Эта модель состоит в следующем: в базарный день на рыночной площади собрались  $N$  продавцов и  $M$  покупателей. У каждого продавца есть одна единица товара (принято говорить о конях); каждый покупатель хочет купить одного коня. С точки зрения покупателей, все продавцы и все кони абсолютно одинаковы, т. е. покупателям всё равно, у кого покупать (значение имеет только цена). Аналогично, с точки зрения продавцов все покупатели одинаковы: продавцам всё равно, кому продавать (опять-таки, значение имеет только цена). Но субъективные оценки товара различаются:

1. Каждый продавец заранее определил для себя цену, дешевле которой он продавать ни в коем случае не будет:  $i$ -й продавец готов продавать коня по цене  $s_i$  и выше.
2. Аналогично, каждый покупатель заранее определил для себя цену, дороже которой он покупать ни в коем случае не будет:  $i$ -й покупатель готов покупать коня по цене  $b_i$  и ниже.

Вдобавок к этим условиям, Е. фон Бём-Баверк справедливо полагает, что каждый продавец будет по возможности запрашивать более высокую цену, в то время как каждый покупатель будет стараться купить как можно дешевле. Далее у Бём-Баверка следуют некоторые рассуждения, которые определяют, какие же конкретно сделки произойдут. По мнению вашего друга, эти рассуждения эквивалентны следующим двум условиям:

3. Все сделки, которые будут совершены, будут совершены по одной цене.
4. Количество продавцов, согласных продавать по этой цене, должно быть равно количеству покупателей и определяет количество сделок.

Тем не менее, эти правила всё ещё не определяют однозначно цены, по которым будут совершены сделки. Ваш друг попросил вас написать

программу, которая определит нижнюю и верхнюю границу диапазона цен, в котором могут быть совершены сделки.

*Примечание:* Вы не ошиблись: вы действительно находитесь на олимпиаде по информатике, а не по экономике :). Поэтому жюри никоим образом не интересуется, не собирается обсуждать и не гарантирует корректность, правильность, реальность и т. д. предложенной модели. Это — задача по информатике, а не по экономике.

### Формат входных данных

В первой строке входного файла находятся два целых числа  $N$  и  $M$  ( $1 \leq N, M \leq 1000$ ) — количество продавцов и покупателей соответственно. На второй строке находятся  $N$  целых чисел  $s_1, s_2, \dots, s_N$ , а на третьей строке —  $M$  целых чисел  $b_1, b_2, \dots, b_M$ ; эти числа положительны и не превосходят 2 000 000 000.

### Формат выходных данных

Если в соответствии с пп. 1—4 может произойти хотя бы одна сделка, выведите в выходной файл два числа: сначала нижнюю границу диапазона, потом — верхнюю. Если не произойдёт ни одной сделки, выведите в выходной файл одну строку 'No solution' (без кавычек).

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
4 3 150 110 200 1000 190 100 140	140 150
1 1 100 90	No solution
2 1 150 150 100	No solution

### Решение

Решение этой задачи при данных ограничениях особой сложности не представляло. Достаточно было просто перебрать все  $N + M$  «пограничных» цен (все  $s_i$  и все  $b_i$ ), и выбрать из них наименьшую и наибольшую, удовлетворяющую условию. Полученное решение требует времени порядка  $(N + M)^2$  и довольно несложно для реализации. Такое решение и приведено в качестве примера.

Заметим, что, отсортировав предварительно цены, можно написать решение, требующее всего  $(N + M) \log(N + M)$  времени. Решения, использующие этот метод, немного сложнее в идейном плане (на самом деле, существуют две весьма различных идеи, использующих сортировку: можно либо сортировать все цены в одном массиве, сохраняя указания, к какому типу, продавцов или покупателей, относится эта цена, либо сортируя их независимо, одни — по возрастанию, а другие — по убыванию), но по сложности реализации ненамного превосходят квадратичное решение. Примеры реализации такого способа можно найти в архиве решений жюри, доступном по адресу <http://olympiads.nnov.ru> (kap\_g.pas, bud\_g.pas, реализующие сортировку всех цен вместе, и hai\_g.pas и tir\_g.pas, сортирующие массивы отдельно).

### Пример правильной программы

```

const maxn=1000;
      maxv=2000000000;
      inf=maxlongint-16;
var n,m:integer;
    ss,bs:array[0..maxn-1]of longint;
    mk:integer;
    minp,maxp:longint;

procedure readin;
var i:integer;
begin
  readln(n,m);
  for i:=0 to n-1 do
    read(ss[i]);
  readln;
  for i:=0 to m-1 do
    read(bs[i]);
  readln;
end;

procedure addp(k:integer; p:longint);
begin
  if mk<k then begin
    mk:=k; minp:=p; maxp:=p;
  end
  else if k=mk then begin
    if minp>p then minp:=p;
    if maxp<p then maxp:=p;
  end;
end;

function getnum(p:longint):integer;
var i:integer;
      s,b:integer;
begin
  s:=0; b:=0;
  for i:=0 to n-1 do
    if p>=ss[i] then inc(s);
  for i:=0 to m-1 do
    if p<=bs[i] then inc(b);
  if s<b then getnum:=s
  else getnum:=b;
end;

procedure solvewriteout;
var i:integer;
    k:integer;
begin
  mk:=0; minp:=inf; maxp:=-inf;
  for i:=0 to n-1 do
    addp(getnum(ss[i]),ss[i]);
  for i:=0 to m-1 do
    addp(getnum(bs[i]),bs[i]);
  if mk<=0 then writeln('No solution')
  else writeln(minp,' ',maxp);
end;

begin
  assign(input,'horses.in');
  reset(input);
  assign(output,'horses.out');
  rewrite(output);
  readin;
  solvewriteout;
end.
```

## Задача 5. Строительство в городе

<i>Входной файл</i>	build.in
<i>Выходной файл</i>	build.out
<i>Ограничение по времени</i>	4 с
<i>Максимальный балл за задачу</i>	100

Новейшие веяния достигли и Солнечного города. Один местный бизнесмен решил построить в городе современный торгово-развлекательный комплекс. Ведущие архитекторы города уже разработали проект, теперь осталось только определить, где он будет размещён. Конечно, предприниматель хотел бы, чтобы комплекс был построен как можно ближе к центру города, но городская администрация оказалась категорически против сноса любых зданий в городе. Поэтому теперь перед предпринимателем встала задача: найти находящийся ближе всего к центру города свободный участок достаточного для строительства размера. Напишите программу, которая решит эту задачу.

Солнечный город, как известно, застроен круглыми домами. Естественно, дома не пересекаются, но некоторые могут касаться. Торгово-развлекательный комплекс тоже будет круглым и тоже не должен пересекаться с уже существующими зданиями (касания допустимы). Расстояние от здания до центра города понимается как расстояние от центра этого здания до центра города.

### Формат входных данных

На первой строке входного файла находятся два числа: целое  $N$  ( $1 \leq N \leq 800$ ) и вещественное  $R$  ( $0 < R \leq 10^6$ ) — количество уже существующих зданий и радиус торгово-развлекательного комплекса.

Далее следуют  $N$  строк, на  $i$ -й из которых находятся три числа  $x_i$ ,  $y_i$  и  $r_i$  — координаты центра и радиус  $i$ -го здания. Координаты не превосходят  $10^6$  по модулю, радиусы положительны и не превосходят  $10^6$ .

Система координат введена таким образом, что центр города имеет координаты  $(0, 0)$ .

### Формат выходных данных

В выходной файл выведите два числа: координаты центра торгово-развлекательного комплекса. Выданная точка должна удовлетворять следующим требованиям:

- 1) для каждого  $i$  расстояние от выданной точки до центра  $i$ -го здания должно быть больше, чем  $r_i + R - 10^{-3}$ ;
- 2) расстояние от выданной точки до начала координат должно отличаться от оптимального не более чем на  $10^{-3}$ .

## Пример

Входной файл	Выходной файл
2 1.0 -1 0 1.0 1 0 1	0 1.7320508075688772

## Решение

В этом разборе под *кругом* будем понимать *открытый* круг, т. е. множество точек, удалённых от центра круга на расстояние, *строго меньшее* радиуса. Таким образом, окружность с тем же центром и радиусом *не будет* иметь общих точек с кругом.

Увеличим радиус каждого здания на  $R$ ; обозначим полученные круги  $o_1, o_2, \dots, o_N$ , а соответствующие окружности —  $\omega_1, \dots, \omega_N$ . Теперь, очевидно, нам необходимо среди всех точек, лежащих вне кругов, найти наиболее близкую к началу координат.

Несложно показать, что искомая точка всегда является одной из следующих:

1. началом координат;
2. точкой какой-нибудь окружности  $\omega_i$ , наиболее близкой к началу координат (на самом деле здесь достаточно рассматривать только такие  $i$ , что круг  $o_i$  содержит начало координат);
3. точкой пересечения двух окружностей.

Не представляет особой сложности перебрать все точки из категорий 1 и 2, проверить, не лежат ли они внутри каких-нибудь кругов, и выбрать из тех, что не лежат, самую близкую к началу координат. Единственным нетривиальным моментом в п. 2 является обработка случая, когда центром некоторой окружности является само начало координат, но в таком случае несложно видеть, что достаточно рассмотреть одну *любую* точку этой окружности.

Обработка точек из категории 3 представляет бóльшую сложность. Действительно, в худшем случае число таких точек может быть порядка  $N^2$ ; на проверку каждой точки необходимо время порядка  $N$  (чтобы проверить, что эта точка лежит вне всех кругов), таким образом, полное время работы простого алгоритма будет порядка  $N^3$ , что не будет укладываться в ограничение времени при данных ограничениях на  $N$ .

Поэтому для обработки точек из категории 3 требуется применить другой способ. Покажем, как можно за время порядка  $N \log N$  при фиксированном  $i$  обработать точки пересечения  $i$ -й окружности со всеми

остальными, т. е. среди точек, лежащих вне всех кругов, выбрать самую ближнюю к началу координат. Пересечём  $i$ -ю окружность со всеми остальными. Выберем на ней некоторую «начальную точку» и отсортируем точки пересечения в порядке обхода окружности начиная с этой точки. При таком обходе каждая точка пересечения будет либо «точкой входа» в соответствующий (пересекающий) круг, либо «точкой выхода» из него, причём тип точки легко определяется при нахождении точек пересечения. Кроме того, при нахождении же точек пересечения можно определить число кругов, накрывающих «начальную точку». Теперь, двигаясь вдоль окружности (т. е. перебирая точки пересечения в отсортированном порядке), можно поддерживать количество  $k$  кругов, внутри которых мы находимся в данный момент. Если до или после прохождения какой-либо точки пересечения  $k$  было/стало равным 0, значит, эта точка лежит вне всех кругов и является кандидатом на ответ. Обойдя всю окружность, мы переберём все точки-кандидаты на ответ. Необходимо ещё заметить, что если несколько точек пересечения совпадают, то сортировать их надо по типу: сначала «выходы», потом «входы».

Применив этот алгоритм для всех  $i$ , мы переберём все кандидаты на ответ из категории 3 за время порядка  $N^2 \log N$ , что вполне допустимо. Осталось среди всех найденных точек выбрать самую близкую к началу координат.

## Пример правильной программы

```
{$n+,r-,q-,s-,i-}
{$M 65520,1024,655360}
const eps=1e-5;arccoseps=1e-10;
var x,y,r:array[1..800] of double;
    n:integer;
    r0,l:double;
    dd,d:array[1..2000] of integer;
    np,nn:integer;
    min,xmin,ymin:double;
    f:text;
    i,j:integer;

procedure solve;
var pp,p:array[1..2000] of double;

function inside(x0,y0:double):boolean;
var i:integer;
begin
  inside:=true;
  for i:=1 to n do if sqr(x0-x[i])+
    sqr(y0-y[i])<r[i]*r[i]-eps then
    exit;
  inside:=false;
end;
```

```
procedure check(x,y:double;
  checkinside:boolean);
begin
  if (not checkinside)
    or (not inside(x,y)) then
  if sqr(x)+sqr(y)<min then begin
    min:=sqr(x)+sqr(y);
    xmin:=x;ymin:=y;
  end;
end;

function arccos(c:double):double;{0..Pi}
var r:double;
begin
  if abs(c)<arccoseps then r:=Pi/2
  else begin
    r:=arctan(sqrt(1-c*c)/abs(c));
    if c<0 then r:=pi-r;
  end;
  arccos:=r;
end;

function arctan2(x,y:double):double;
```

```

var r:double;
begin
  if abs(x)<eps then
    if y<0 then r:=3*pi/2
    else r:=pi/2
  else if x>0 then
    r:=arctan(y/x)
  else r:=pi+arctan(y/x);
  if r<0 then r:=r+2*pi;
  arctan2:=r;
end;

function norm(a:double):double;
begin
  if a>2*pi then norm:=a-2*pi
  else if a<0 then norm:=a+2*pi
  else norm:=a;
end;

procedure cross(i,j:integer);
var a,b,dd:double;
begin
  dd:=sqrt(sqr(x[i]-x[j])+sqr(y[i]-y[j]));
  if dd>r[i]+r[j]-eps then exit;
  a:=arccos((r[i]*r[i]+dd*dd-
            r[j]*r[j])/2/r[i]/dd);
  b:=arctan2(x[j]-x[i],y[j]-y[i]);
  inc(np);p[np]:=norm(b+a);d[np]:=-1;
  inc(np);p[np]:=norm(b-a);d[np]:=1;
  if (b<a)or(2*pi-b<a) then inc(nn);
end;

function less(i,j:integer):boolean;
begin
  less:=(p[i]<p[j]-eps) or
  ((abs(p[i]-p[j])<eps)and(d[i]<d[j]));
end;

procedure sort(l,r:integer);
var i,i1,i2,o,tt:integer;
    t:double;
begin
  if l>r then exit;
  o:=(l+r) shr 1;
  sort(l,o);sort(o+1,r);
  i1:=l;i2:=o+1;
  for i:=1 to r do
    if (i2>r)or((i1<=o)and
      (less(i1,i2))) then begin
      pp[i]:=p[i1];dd[i]:=d[i1];

```

```

      inc(i1);
    end else begin
      pp[i]:=p[i2];dd[i]:=d[i2];
      inc(i2);
    end;
  for i:=1 to r do begin
    p[i]:=pp[i];d[i]:=dd[i];
  end;
end;

begin
  assign(f,'build.in');reset(f);
  read(f,n,r0);
  for i:=1 to n do begin
    read(f,x[i],y[i],r[i]);
    r[i]:=r[i]+r0;
  end;
  close(f);
  min:=1e20;
  check(0,0,true);
  for i:=1 to n do begin
    if abs(x[i]+abs(y[i])>0) then begin
      l:=sqrt(sqr(x[i])+sqr(y[i]));
      check(x[i]-x[i]/l*r[i],
            y[i]-y[i]/l*r[i],true);
      end else check(r[i],0,true);
      nn:=0;
      np:=0;
      for j:=1 to n do if i<j then
        cross(i,j);
      sort(1,np);
      for j:=1 to np do begin
        if nn=0 then
          check(x[i]+r[i]*cos(p[j]),
                y[i]+r[i]*sin(p[j]),false);
          nn:=nn+d[j];
        if nn=0 then
          check(x[i]+r[i]*cos(p[j]),
                y[i]+r[i]*sin(p[j]),false);
        end;
      end;
  assign(f,'build.out');rewrite(f);
  writeln(f,xmin:20:20,' ',ymin:20:20);
  close(f);
end;

begin
  solve;
end.

```

## Задача 6. Кофе

<i>Входной файл</i>	<code>coffee.in</code>
<i>Выходной файл</i>	<code>coffee.out</code>
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

В Научно-образовательном центре ИПФ РАН недавно заново покрасили стены. Всё было бы ничего, если бы на этаж ниже не стоял кофейный аппарат: особую опасность для недавно покрашенных стен представляют школьники, пьющие кофе, поскольку иногда они разливают кофе и он забрызгивает стены, находящиеся от школьника на расстоянии, меньшем некоторого заданного  $R$  (если расстояние до стены равно  $R$ , она не пачкается). Для борьбы с этой напастью в одной из секретных лабораторий ИПФ РАН разрабатывается система, которая в момент разливания школьником кофе регистрирует его координаты  $(x, y)$  и радиус разлёта кофе  $R$ . Предполагается, что она будет устанавливаться во всех свежепокрашенных комнатах, а данные наблюдений будут записываться в единый журнал. Каждая комната считается прямоугольником ненулевой площади со сторонами, параллельными осям координат.

Вашей задачей в данном проекте является разработка кода, который будет определять, какие случаи разливания кофе, зафиксированные в журнале, завершились запачкиванием стен.

### Формат входных данных

Во входном файле приведено содержание журнала разливания кофе. В первой строке указано число записей в журнале  $N$  ( $1 \leq N \leq 20$ ). Во второй и последующих строках приведены сами записи, по одной записи на строку. Каждая запись представляет собой 7 чисел  $x_1, y_1, x_2, y_2, R, x, y$ , разделённых пробелами. Здесь  $(x_1, y_1), (x_2, y_2)$  — координаты двух противоположных углов комнаты, в которой зафиксировано данное разливание кофе,  $R$  — радиус разлёта кофе,  $(x, y)$  — координаты точки внутри комнаты, в которой кофе был разлит. Все координаты во входном файле целые и не превосходят по модулю 10 000, радиус  $R$  тоже целый,  $0 \leq R \leq 10\,000$ .

### Формат выходных данных

В выходом файле должно быть  $N$  строк. На  $i$ -й строке должно быть выведено 'yes', если при  $i$ -м разливании кофе испачкало стену, иначе должно быть выведено 'no'.

**Пример**

<i>Входной файл</i>	<i>Выходной файл</i>
3	no
-2 1 10 9 1 0 2	no
3 5 16 -3 2 10 0	yes
3 -6 5 20 2 4 13	

**Решение**

Это одна из самых простых из предлагавшихся задач. Поскольку условия гарантируют, что школьник стоит внутри комнаты, для решения достаточно заметить, что кофе пачкает стену тогда и только тогда, когда расстояние от школьника до этой стены строго меньше  $R$ . Расстояние до стены определяется как модуль разности соответствующих координат школьника и стены:

$$|x - x_1|, \quad |x - x_2|, \quad |y - y_1|, \quad |y - y_2|.$$

Разливание кофе завершается запачкиванием стен, если расстояние до какой-нибудь из стен меньше  $R$ , в противном случае стены остаются чистыми. Таким образом, ответом для каждой из строк журнала является значение логического выражения

$$(|x - x_1| < R) \vee (|x - x_2| < R) \vee (|y - y_1| < R) \vee (|y - y_2| < R),$$

где символом  $\vee$  обозначена операция «логическое ИЛИ» (дизъюнкция).

**Пример правильной программы**

```
var n:integer;
    x0,x1,y0,y1:integer;
    r,x,y:integer;

const
  res:array[boolean]of string[3]=
    ('no','yes');

begin
  assign(input,'coffee.in');
  reset(input);
  readln(n);
```

```
assign(output,'coffee.out');
rewrite(output);
while n>0 do begin
  readln(x0,y0,x1,y1,r,x,y);
  writeln(res[(abs(x0-x)<r)or
    (abs(x1-x)<r)or
    (abs(y0-y)<r)or
    (abs(y1-y)<r)]);
  dec(n);
end;
end.
```

### Результаты III Нижегородской городской олимпиады школьников по информатике

	Школа	Кл.	1	2	3	4	5	6	= Дипл.
1. P86: Елифанов Владислав	лиц. 40	9	100	100	44	88	23	100	455 I
2. P10: Побуринная Оксана	гимн. 2, г. Саров	10	100	100		84	68	100	452 I
3. P81: Разенштейн Илья	лиц. 40	11	43	100	18	100	27	100	388 I
4. P35: Погорелов Дмитрий	шк. 165	9	31	100		54		100	285 I
5. P20: Петровичев Максим	шк. 14, г. Балахна	11	20	80		79	0	100	279 I
6. P93: Крылова Ирина	лиц. 87	9	32	100	4	15		100	251 II
7. P11: Огнева Дарья	лиц. 15, г. Саров	9	31	100		15		100	246 II
8. P87: Фадеев Сергей	лиц. 40	7	28	100	8	7		100	243 II
9. P05: Банин Максим	шк. 117	11	28	100		8	5	100	241 II
10. P61: Савинов Сергей	шк. 121	11	18	90	0	26	5	100	239 II
11. P37: Низовцев Сергей	шк. 165	9	100	25			9	100	234 II
12. P41: Чистяков Андрей	лиц. 38	9		95	2	33		100	230 II
13. P02: Хусаинов Тимур	шк. 85	11	18	100		10		100	228 II
13. P15: Огнева Мария	гимн. 2, г. Саров	7	28	90		10		100	228 II
15. P82: Булатов Алексей	лиц. 40	10		50		74		100	224 III
16. P68: Февральских Сергей	шк. 121	11	5	100		15		100	220 III
17. P88: Костров Александр	шк. 149	11	10	100		7		100	217 III
18. P39: Громов Михаил	шк. 36	9		95		20		100	215 III
19. P83: Рябкин Николай	лиц. 40	10		90		22		100	212 III
20. P07: Мальженков Александр	шк. 82	11	0	100		10		100	210 III
20. P16: Дубатовка Алина	лиц. 15, г. Саров	7	0	100		10	0	100	210 III
22. P14: Николайчук Дарья	лиц. 15, г. Саров	7		90		17		100	207 III
23. P48: Царев Михаил	лиц. 38	11	0	100		5		100	205 III
24. P47: Мартынов Игорь	шк. 18	11	0	100	2	0		100	202 III
24. P84: Розенштейн Борис	лиц. 40	10	0	100	2	0	0	100	202 III
26. P92: Рожнецев Алексей	шк. 70	11		100				100	200 III
27. P40: Бычков Илья	шк. 63	10	28	70		0	0	100	198 III
28. P18: Шашкин Юрий	шк. 14, г. Балахна	10	0	80	16	0	0	100	196 III
29. P38: Вилов Сергей	шк. 165	9		80		0		100	180 III

30.	Р67: Герасимов Сергей	гимн. 50	11	0	10	7	51	5	100	173	III
31.	Р03: Крот Александр	шк. 85	9		50		10	5	100	165	III
32.	Р85: Корольков Михаил	лиц. 40	10	0	35		18		100	153	III
33.	Р31: Самойлыч Максим	шк. 165	11	35	90		19		8	152	III
34.	Р64: Белоножкин Петр	гимн. 2	11		45				100	145	
35.	Р13: Курагова Наталья	лиц. 15, г. Саров	7	28	100		15		0	143	
36.	Р06: Калинин Арсений	шк. 82	11	5	30		0	0	100	135	
37.	Р09: Кочеганов Виктор	шк. 82	11		25				100	125	
38.	Р69: Дергунов Олег	шк. 121	11	5	100		14	3	0	122	
39.	Р63: Грачев Денис	гимн. 2	10		0		14		100	114	
40.	Р12: Анисимова Светлана	гимн. 2, г. Саров	8		10		0		100	110	
41.	Р33: Грачев Андрей	шк. 36	11		0		9	0	100	109	
42.	Р17: Вадимов Василий	шк. 14, г. Балахна	9	0	100			5	0	105	
43.	Р21: Петровичев Александр	шк. 14, г. Балахна	10	43	50		7	0	0	100	
43.	Р52: Трущенков Михаил	шк. 91	11		100		0			100	
45.	Р34: Брандт Андрей	шк. 165	11	0	85		0	8	0	93	
46.	Р62: Дегтярев Владимир	шк. 121	11	0	60	4	7	5	6	82	
47.	Р49: Соломин Сергей	лиц. 38	11	0	50		13		4	67	
48.	Р36: Тихонов Андрей	шк. 165	9	32	25	2			0	59	
49.	Р51: Кочетунов Антон	шк. 180	10	0	0		0		50	50	
50.	Р43: Камаев Артем	лиц. 38	9		10				25	35	
51.	Р94: Борзая Екатерина	лиц. 87	9		20		13		0	33	
52.	Р66: Бовкун Леонид	гимн. 2	10	0	20		0		0	20	

В таблице не указаны участники, набравшие менее 10 баллов (13 человек).

# IV городская олимпиада школьников по информатике 28 января 2008 г.

## Авторы задач

Бударагин Д. И., студент 2 курса ВШОПФ ННГУ;  
Калинин П. А., студент 6 курса ВШОПФ ННГУ;  
Круглов А. А., аспирант ИПФ РАН;  
Лелюх В. Д., старший преподаватель ННГУ;  
Разенштейн И. П., студент 1 курса мехмата МГУ;  
Тимушев Р. И., студент 5 курса ВШОПФ ННГУ;  
Хаймович И. М., студент 5 курса ВШОПФ ННГУ.

## Задача 1. Счастливые билеты — 2

<i>Входной файл</i>	<code>tickets.in</code>
<i>Выходной файл</i>	<code>tickets.out</code>
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Ваш друг всё ещё работает главным экономистом в фирме, владеющей несколькими маршрутками. Недавно в его фирме был проведён ряд мероприятий по защите билетов на маршрутки от подделок. Среди прочих мер была изменена система номеров билетов. Теперь на каждом билете напечатано  $N$  чисел, каждое в диапазоне от 1 до  $N$ ; т. е. имеется всего  $N^N$  различных вариантов билета.

В результате автобусы этой фирмы стали пользоваться ещё большей популярностью, чем раньше. С целью получить новый билет в качестве сувенира люди стали ездить в этих маршрутках, даже если им надо было ехать в другую сторону. Даже по выходным маршрутки стали ходить битком набитыми. Как главный экономист фирмы, ваш друг решил, что такая популярность маршруток создаёт возможность получения дополнительной прибыли. В результате были повышены цены на билеты, что, правда, не привело к серьёзному уменьшению пассажиропотока.

Кроме того, по сообщениям кондукторов маршруток, среди пассажиров началась борьба за счастливые билеты. Правда, стандартные

способы определения «счастливости» билета не работают при новой системе номеров. Поэтому, специально по указанию вашего друга, кондукторы маршруток провели исследование, которое показало, что пассажиры придумали новое определение счастливого билета. А именно, если обозначить числа, напечатанные на билете, как  $m(1), m(2), \dots, m(N)$ , то билет счастливый тогда и только тогда, когда  $m(m(i)) = m(i)$  для каждого  $i$  от 1 до  $N$ .

Например, при  $N = 2$  из 4 возможных билетов три являются счастливыми, и только билет 2 1 — не счастливый (т. к., например,  $m(m(2)) = m(1) = 2 \neq m(2) = 1$ ); при  $N = 4$ , например, билеты 1 1 3 1 и 1 2 3 4 являются счастливыми, а 2 2 2 3 — нет (т. к.  $m(m(4)) = m(3) = 2 \neq m(4) = 3$ ).

Ваш друг решил, что можно получить дополнительный доход, продавая счастливые билеты с наценкой. Теперь для составления экономического плана ему нужно знать, сколько существует счастливых билетов при данном  $N$ . Помогите ему.

### Формат входных данных

Во входном файле находится одно число  $N$  ( $1 \leq N \leq 30$ ).

### Формат выходных данных

В выходной файл выведите одно число — количество возможных счастливых билетов при данном  $N$  без ведущих нулей.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
2	3

### Решение

Это была простая задача на комбинаторику.

Фактически, требовалось найти число функций  $m: \{1, \dots, N\} \rightarrow \{1, \dots, N\}$  таких, что  $m(m(x)) = m(x)$  для любого  $x \in \{1, \dots, N\}$ . Рассмотрим область значений любой из искомым функций. Пусть это будет множество  $\Lambda$ . Тогда  $m(x) = x$  для любого  $x \in \Lambda$ , а, кроме того,  $m(x) \in \Lambda$  для любого  $x \notin \Lambda$ . Пусть размер  $\Lambda$  равен  $k$ . Тогда возможных областей значения существует  $C_n^k$  — количество способов выбрать  $k$  предметов из  $n$ , причём порядок выбранных предметов не важен. А значения  $m(x)$  для  $x \notin \Lambda$  мы можем назначить  $k^{n-k}$  способами. В итоге ответ имеет следующий вид:  $\sum_{k=1}^n C_n^k k^{n-k}$ .

Осталось две проблемы: научиться считать  $C_n^k$  и реализовать «длинную арифметику».  $C_n^k$  лучше всего считать с помощью треугольника Паскаля:  $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$ ,  $C_n^n = C_n^0 = 1$ .

Так как ответ в задаче не превосходит 30 цифр, то проще всего было хранить массив десятичных цифр и реализовать сложение двух длинных чисел и умножение длинного числа на короткого «в столбик».

### Пример правильной программы

```

const base=10000;
      lgbase=4;
type tlong=array [0..10] of longint;
var n,i,j:longint;
      c:array [0..30] of tlong;
      a,t:tlong;
      ts:string;

procedure add(var a,b:tlong);
var os,i:longint;
begin
  if b[0]>a[0] then
    a[0]:=b[0];
  os:=0;
  for i:=1 to a[0] do begin
    inc(a[i],b[i]+os);
    os:=0;
    if a[i]>base then begin
      dec(a[i],base);
      os:=1;
    end;
  end;
  if os=1 then begin
    inc(a[0]);
    a[a[0]]:=1;
  end;
end;

procedure mul(var a:tlong; b:longint);
var os,i:longint;
begin
  os:=0;
  for i:=1 to a[0] do begin
    inc(os,a[i]*b);
    a[i]:=os mod base;
    os:=os div base;
  end;
end;

while os>0 do begin
  inc(a[0]);
  a[a[0]]:=os mod base;
  os:=os div base;
end;

begin
  assign(input,'tickets.in');
  reset(input);
  assign(output,'tickets.out');
  rewrite(output);
  read(n);
  fillchar(c,sizeof(c),0);
  c[0][0]:=1;
  c[0][1]:=1;
  for i:=1 to n do
    for j:=i downto 1 do
      add(c[j],c[j-1]);
  fillchar(a,sizeof(a),0);
  for i:=1 to n do begin
    t:=c[i];
    for j:=1 to n-i do
      mul(t,i);
    add(a,t);
  end;
  write(a[a[0]]);
  for i:=a[0]-1 downto 1 do begin
    str(a[i],ts);
    for j:=length(ts)+1 to lgbase do
      write(0);
    write(ts);
  end;
  writeln;
  close(input); close(output);
end.

```

## Задача 2. Коды городов

<i>Входной файл</i>	citycode.in
<i>Выходной файл</i>	citycode.out
<i>Ограничение по времени</i>	5 с
<i>Максимальный балл за задачу</i>	100

Как известно, система телефонных номеров в России устроена следу-

ющим образом. Все телефонные номера имеют одну и ту же длину —  $M$  цифр (на самом деле  $M = 10$ , но в этой задаче ваша программа должна будет работать для различных  $M$ , не превосходящих 9). Каждый номер состоит из двух частей: несколько первых цифр номера составляют код города, остальные цифры определяют номер абонента в пределах этого города.

Телефонная сеть в России постоянно развивается, коды городов регулярно меняются, при этом, конечно, коды различных городов обязательно различны. В остальном коды могут быть совершенно произвольными, в том числе код одного города может быть началом кода другого (например, код Нижнего Новгорода — 831, а код Сарова — 83130) и т. п. В случае, если коды *нескольких* городов являются началом  $M$ -значного номера абонента, кодом города этого номера считается наидлиннейший из таких кодов. Например, если есть всего четыре города: Нижний Новгород, Балахна, Дзержинск и Саров с кодами, соответственно, 831, 83144, 8313 и 83130, и  $M = 9$ , то: телефоны 831123456 и 831412345 — телефоны Нижнего Новгорода, 831312345 — телефон Дзержинска, 831301234 — Сарова, а 813441234 — Балахны.

В этой задаче мы не будем учитывать различные другие ограничения на телефонные номера, существующие в реальности (например, в реальности никакой номер абонента в городе не может начинаться на 03, т. к. 03 — это телефон скорой помощи). Мы будем считать, что любой из  $10^M$  возможных  $M$ -значных номеров является допустимым телефонным номером.

При подобной системе нетривиальной задачей становится задача определения, сколько всего телефонных номеров может быть выдано в каждом городе. Например, в приведённом выше примере в Балахне и Сарове могут быть выданы 10 000 телефонных номеров (все четырёхзначные номера), в Дзержинске могут быть выданы 90 000 телефонных номеров — все пятизначные телефонные номера, кроме начинающихся на ноль, а в Нижнем Новгороде могут быть выданы 890 000 номеров — все шестизначные номера, кроме тех, которые начинаются на цифру 3, а также на последовательность 44.

Напишите программу, которая решит эту задачу для произвольного набора кодов городов.

### Формат входных данных

На первой строке входного файла находятся два числа  $N$  и  $M$  — количество городов и длина полного телефонного номера ( $1 \leq N \leq 100\,000$ ,  $1 \leq M \leq 9$ ). Далее следуют  $N$  строк, в каждой из которых находится

код очередного города и название города, между которыми находится ровно один пробел. Название города состоит только из заглавных и маленьких латинских букв и не превышает по длине 20 символов. Гарантируется, что в каждой из этих строк входного файла будет ровно один пробел — между кодом города и его названием.

Города во входном файле отсортированы по алфавитному порядку кодов. Гарантируется, что никакие два кода и никакие два названия города не совпадают.

### Формат выходных данных

В выходной файл выведите  $N$  строк. В каждой строке выведите название города и количество телефонных номеров, которые можно выдать в этом городе. Отделяйте название города и количество номеров как минимум одним пробелом. Вы также можете выводить лишние пробелы в начало и конец каждой строки; они будут игнорироваться.

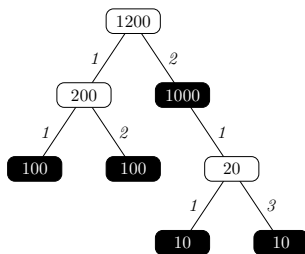
Города можете выводить в произвольном порядке.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
4 9	Sarov 10000
831 NizhnyNovgorod	NizhnyNovgorod 890000
8313 Dzerzhinsk	Balakhna 10000
83130 Sarov	Dzerzhinsk 90000
83144 Balakhna	

### Решение

Эта задача могла бы быть легко решена с помощью структуры данных, называемой *бор* (англ. *trie*, также употребляются термины *луч* и *нагруженное дерево*). Бор — это дерево с корнем, каждое ребро которого помечено некоторым символом; при этом из каждой вершины в её дочерние вершины выходит не более одного ребра, помеченного каждым символом. В таком случае каждой вершине дерева однозначно соответствует строка, которая читается по рёбрам при движении от корня к этой вершине (при этом корню соответствует пустая строка), а каждой возможной строке — не более одной вершины. Бор для данного набора строк — это



минимальный бор, в котором каждой строке из данного набора соответствует вершина.

Покажем, как с помощью такой структуры можно было решить задачу. Построим бор для набора кодов городов, заданных во входном файле. Например, бор для набора кодов 11, 12, 2, 211 и 213 показан на рисунке. Будем в каждой вершине бора хранить название города с соответствующим этой вершине кодом (или пустую строку, если такого города нет), а также количество номеров телефонов, которые заняты в поддереве с корнем в этой вершине (т. е. сумму количеств номеров, приходящихся на все города, попавшие в поддерево с корнем в этой вершине — обозначим её *used*). На рисунке значения *used* для  $M = 4$  приведены в вершинах; чёрная вершина обозначает вершину, которой соответствует некоторый город, белая — вершину, которой никакой город не соответствует.

Такой бор можно было бы хранить в динамической памяти с помощью следующей структуры:

```
type pnode=~tnode;
  tnode=record
    name:string; {название города}
    used:integer; {кол-во использованных номеров}
    next:array['0'..'9'] of pnode; {дети данной вершины}
  end;
```

Бор для данного набора строк строится просто: начиная с дерева, содержащего только корень, мы последовательно добавляем строки, создавая по пути отсутствующие ребра.

Покажем, как можно вычислить значения *used* для всех вершин бора за линейное по количеству вершин время. Напишем рекурсивную процедуру подсчёта значения *used* и решения задачи для некоторой вершины. Сначала она запустит себя для дочерних вершин. После этого:

- Если данной вершине не соответствует никакой город, то просто *used* для данной вершины будет равен сумме *used* её детей;
- Если же данной вершине соответствует некоторый город, то, во-первых, в данной вершине использованы *все* номера (либо в этом городе, либо в городах-потомках данной вершины), т. е. *used* для данной вершины равно  $10^{M-h}$ , где  $h$  — глубина этой вершины (расстояние от неё до корня). Во-вторых, в городе, соответствующем

этой вершине, количество возможных номеров равно  $10^{M-h}$  минус сумма всех *used* детей этой вершины — это значение вместе с названием города можно сразу выводить в выходной файл.

Построив бор и запустив эту процедуру от корня, мы решили бы задачу. Единственная проблема — при данных ограничениях бор не поместится даже в динамической памяти.

Эта проблема в этой задаче решается благодаря тому, что коды во входном файле расположены по алфавиту. В таком случае можно хранить только часть дерева по пути от корня до текущей вершины: будем хранить стек значений *used*, стек пометок на рёбрах (фактически, код текущей вершины) *code*, и стек названий городов *name*; кроме того, храним текущую глубину *len*. Изначально все стеки пусты, *len* = 0. Далее, при чтении очередного кода нам надо подняться на несколько (возможно, ноль) рёбер и после этого спуститься на несколько (возможно, ноль) рёбер. При подъёме будем корректировать значения *used* в текущих вершинах, а также выводить ответы для городов, соответствующих текущим вершинам. При спуске будем инициализировать все значения. После завершения чтения входного файла надо не забыть вывести ответы для городов, оставшихся в стеке, корректируя *used* по мере подъёма по стеку.

## Пример правильной программы

```
var f,g:text;
    code:array[1..10] of char;
    used:array[0..10] of longint;
    name:array[1..10] of string;
    len,maxlen:integer;
    n:longint;
    cc,cn:string;
    i,j,k:longint;
    n0:array[0..9] of longint;
    s:string;

procedure parse(s:string;
    var code,name:string);
var i:integer;
    was:boolean;
begin
    code:= ''; name:= '';
    was:=false;
    for i:=1 to length(s) do begin
        if not was then begin
            if s[i]=' ' then was:=true
            else code:=code+s[i];
        end
        else name:=name+s[i];
    end;
end;
```

```
end;

begin
    n0[0]:=1;
    for i:=1 to 9 do
        n0[i]:=n0[i-1]*10;
    assign(f,'citycode.in');reset(f);
    assign(g,'citycode.out');rewrite(g);
    readln(f,n,maxlen);
    fillchar(code,sizeof(code),#0);
    len:=0;
    for i:=1 to n do begin
        readln(f,s); parse(s,cc,cn);
        j:=1;
        while (j<=len)and(code[j]=cc[j]) do
            inc(j);
        {теперь j --
        первая различающаяся буква ИЛИ j>len}
        for k:=len downto j do begin
            if name[k]<>' ' then begin
                writeln(g,name[k],' ',
                    n0[maxlen-k]-used[k]);
                used[k-1]:=used[k-1]+n0[maxlen-k];
            end
        end
    end;
end;
```

```

used[k-1]:=used[k-1]+used[k];
end;
for k:=j to length(cc) do begin
  code[k]:=cc[k];
  name[k]:='';
  used[k]:=0;
end;
len:=length(cc);
name[len]:=cn;
end;

```

```

for k:=len downto 1 do begin
  if name[k]<>' ' then begin
    writeln(g,name[k],', ',
      n0[maxlen-k]-used[k]);
    used[k-1]:=used[k-1]+n0[maxlen-k];
  end
  else used[k-1]:=used[k-1]+used[k];
end;
close(g);
end.

```

## Задача 3. Собрать квадрат

<i>Входной файл</i>	square.in
<i>Выходной файл</i>	square.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Вы участвуете в проекте по исследованию интеллекта обезьян. Ваш научный руководитель, доктор биологических наук профессор О. Б. Ломов предложил новый тест для определения умственных способностей орангутанов. Обезьяне выдаётся набор из  $N$  палочек различной длины, и она должна составить из них квадрат. Это оказалось не самой простой задачей для ваших подопечных, хотя некоторые особи могут справиться с ним при  $N$ , достигающих 10. Но самой большой сенсацией стали неожиданные успехи орангутана Миши, который научился решать эту задачу даже при очень больших  $N$  — порядка нескольких сотен. Правда, решает он её весьма своеобразно. Он выбирает из всех палочек четыре и составляет из них квадрат.

Последнее время вы проводите с Мишей много экспериментов. К сожалению, не все из них удачны: из некоторых наборов палочек Миша не может собрать квадрат. После многочисленных неудачных попыток заставить Мишу решить задачу вы предположили, что на таких наборах палочек задача неразрешима. К сожалению, проверить её разрешимость оказалось довольно сложно, и вы решили написать программу, которая вам поможет.

### Формат входных данных

На первой строке входного файла находится одно число  $N$  — количество палочек, которые вы предложили Мише ( $1 \leq N \leq 1000$ ). На второй строке находятся  $N$  натуральных чисел, не превосходящих 1000 — длины палочек.

## Формат выходных данных

Если задача разрешима, т. е. из данных палочек можно выбрать четыре, из которых можно собрать квадрат, то выведите в выходной файл одно число — длину стороны какого-нибудь квадрата, который можно собрать. Если решения не существует, выведите в выходной файл одно число  $-1$ .

## Пример

<i>Входной файл</i>	<i>Выходной файл</i>
6 3 3 2 3 2 3	3
6 1 3 2 3 2 3	-1
5 1 1 2 2 2	-1

## Решение

Существует несколько решений этой задачи. Можно было перебрать четыре номера палочек, из которых мы будем пытаться собрать квадрат, и проверять, что их длины равны. Такой алгоритм работает за  $O(N^4)$  и не проходит полный набор тестов.

Для улучшения алгоритма можно заметить, что, если мы зафиксировали длину стороны квадрата, то легко за  $O(N)$  проверить, можно ли собрать такой квадрат — достаточно перебрать все палочки и подсчитать, сколько из них нужной длины. Если их не меньше 4, то можно, иначе нельзя. Получится алгоритм, который работает за  $O(N^2)$  и набирает максимальный балл; такое решение и приведено в качестве примера.

Получить решение, работающее за  $O(N \log N)$  или за  $O(N + K)$ , где  $K$  — максимальная длина палочки, можно было, отсортировав длины палочек, а потом заметив, что палочки, которые образуют квадрат, стоят в отсортированном массиве подряд. Примеры решений, использующие этот подход, можно скачать в архиве решений жюри с сайта <http://olympiads.nnov.ru>.

Отметим также следующий интересный факт. Пусть, как и в условии задачи, длины всех палочек — натуральные числа, не превосходящие  $K$ . Тогда существует решение задачи, работающее за  $O(K)$ , т. е. не требующее *никаких* ограничений на  $N$  и укладывающееся в ограничение времени при сколь угодно больших  $N$ . Действительно, несложно

видеть, что при  $N \geq 3K + 1$  задача *всегда* имеет решение (если решения нет, то для каждой возможной длины соответствующих палочек не более трёх, т.е. всего палочек не более  $3K$ ). Поэтому при  $N \geq 3K + 1$  достаточно считать из входного файла лишь первые  $3K + 1$  длины и решить задачу для них; решение точно найдется. (Если же  $N < 3K + 1$ , то, конечно, надо обработать все палочки, которые есть.)<sup>1</sup>

## Пример правильной программы

```

var
  n,i,cnt,j:longint;
  a:array [1..1000] of longint;

begin
  assign(input,'square.in');
  reset(input);
  assign(output,'square.out');
  rewrite(output);
  read(n);
  for i:=1 to n do
    read(a[i]);
  for i:=1 to n do begin
    cnt:=0;
    for j:=1 to n do
      if a[j]=a[i] then
        inc(cnt);
    if cnt>=4 then begin
      writeln(a[i]);
      close(input);
      close(output);
      halt;
    end;
  end;
  writeln(-1);
  close(input);
  close(output);
end.
```

## Задача 4. Мёртвый код

<i>Входной файл</i>	cover.in
<i>Выходной файл</i>	cover.out
<i>Ограничение по времени</i>	2 с
<i>Максимальный балл за задачу</i>	100

При разработке крупных программных продуктов иногда сложно уследить за всеми ветвями алгоритма. Это может привести к тому, что некоторые куски кода никогда не будут исполняться. Например, в следующем отрывке кода на языке С инструкция `printf` никогда (ни при каких значениях переменных) не будет исполнена:

```

if (a<b) {
  if (b<a) {
    printf("Никогда не будет исполнено\n");
  }
}
```

<sup>1</sup> На самом деле, утверждение, что это решение работает за  $O(K)$ , неверно, т.к. еще требуется время на считывание  $N$  и сравнение его с  $3K + 1$ ; более строго можно оценить время работы этого решения как  $O(K + \log N)$ . Если бы во входном файле не задавалось бы  $N$ , а просто были бы перечислены длины всех палочек, то решение работало бы за  $O(K)$ .

Перед вами стоит задача автоматизации процесса выявления недостижимых участков кода. Конечно, в наиболее общем случае эта задача неразрешима, поэтому мы ограничимся программами очень простой структуры.

### Формат входных данных

Во входном файле находится фрагмент программы, написанной на языке C. Этот фрагмент может содержать только операторы `if`, причём единственным допустимым условием является сравнение значений двух переменных. В качестве оператора сравнения может выступать только оператор «меньше» (`<`). В качестве имён переменных могут быть использованы только одиночные строчные буквы латинского алфавита.

Для простоты строго зафиксируем разбиение программы на строки: каждая строка входного файла может начинаться с произвольного числа пробелов, за которыми следует одна из следующих последовательностей символов:

- `'if □(a<b) □{'` (вместо `a` и `b`, естественно, могут стоять другие имена переменных; `□` — пробел)
- `'}'`

Размер входного файла не превышает 100 Кб. Глубина вложенности операторов `if` не превышает 1000. Длина каждой строки входного файла не превышает 200 символов.

### Формат выходных данных

Выходной файл должен содержать ту же самую программу, что и входной, но недостижимые блоки кода должны быть отмечены комментарием. Недостижимыми блоками считаются такие, код в которых не будет исполняться ни при каких значениях переменных. В начале каждого такого блока, сразу после открывающей фигурной скобки, должен находиться комментарий `'/*DEAD*/'`. Блоки, находящиеся внутри другого недостижимого блока, помечать не следует.

Допускается добавление и удаление пробелов в любое место/из любого места выходного файла.

**Пример**

<i>Входной файл</i>	<i>Выходной файл</i>
<pre> if (a&lt;b) {   if (b&lt;c) {     if (c&lt;a) {       if (x&lt;y) {       }     }   } } if (x&lt;y) { } } if (b&lt;a) {   if (c&lt;b) {   } } } </pre>	<pre> if ( a &lt;b) {   if(b&lt; c) {     if (c&lt;a) { /*DEA D*/       if (x&lt;y) {       }     }   } } i f (x&lt;y) { } } if(b&lt;a) { if(c&lt;b){ } } } </pre>

**Решение**

Для решения этой задачи удобно переформулировать условие в терминах теории графов. Сопоставим нашей программе динамически изменяющийся ориентированный граф. Вершинами в нем будут переменные, а рёбрами — наложенные на эти переменные условия. Точнее говоря, ребро из вершины  $a$  в вершину  $b$  будет обозначать, что в соответствии с операторами `if` значение переменной  $a$  должно быть меньше значения переменной  $b$ . Очевидно, что мёртвому коду в этой интерпретации соответствуют циклы в графе.

Будем читать входной файл построчно. Пусть прочитана строка, содержащая оператор `if (a<b) {`. Если в графе существует ребро  $b \rightarrow a$ , то условие  $a < b$  никогда не будет выполнено и, следовательно, код внутри данного оператора `if` недостижим. В противном случае, добавим в граф все ребра  $x \rightarrow y$  такие, что ребра  $x \rightarrow a$  и  $b \rightarrow y$  уже существуют (будем изначально предполагать, что ребра  $c \rightarrow c$  существуют для всех  $c$ ). Очевидно, что тем самым мы отразим в графе все возникшие ограничения на значения переменных. Когда из файла будет прочитана строка `}`, необходимо удалить ребра, добавленные соответствующим оператором `if`. Для этого достаточно сохранять список добавленных на каждом шаге рёбер в отдельном массиве.

Алгоритм требует аккуратной реализации; в частности, как показали результаты олимпиады, сложность может представлять коррект-

ная обработка примеров вида `if (a<a)...`, а также вида `if (a<b){ if (a<b) {} if (b<a) {} }`.

## Пример правильной программы

```

const MAX = 26;
      MAXDEEP = 1000;
var m:array['a'..'z', 'a'..'z'] of boolean;
    q:array[1..MAX*MAX,1..2] of char;
    steps:array[0..MAXDEEP] of integer;
    step:integer;
    dead:integer;

procedure setM(i,j:char);
begin
  if not m[i,j] then begin
    m[i,j]:=true;
    inc(steps[step]);
    q[steps[step],1]:=i;
    q[steps[step],2]:=j;
  end;
end;

procedure go(i,j:char);
var p,q:char;
begin
  if dead>0 then begin
    inc(dead); exit;
  end;
  if m[j,i] then begin
    write('/*DEAD*/*');
    dead:=1; exit;
  end;
  inc(step);
  steps[step]:=steps[step-1];
  if not m[i,j] then begin
    setM(i,j);
    for p:='a' to 'z' do
      for q:='a' to 'z' do
        if m[p,i] and m[j,q] then setM(p,q);
      end;
    end;
  end;
  procedure goBack;
  var i:integer;
  begin
    if dead>0 then begin
      dec(dead); exit;
    end;
    for i:=steps[step-1]+1 to steps[step] do
      m[q[i,1],q[i,2]]:=false;
    dec(step);
  end;
  var s:string;
      i:char;
  begin
    assign(input,'cover.in');
    reset(input);
    assign(output,'cover.out');
    rewrite(output);
    for i:='a' to 'z' do m[i,i]:=true;
    while not seekeof(input) do begin
      readln(s); write(s);
      if s[1]='i' then go(s[5],s[7])
      else goBack;
      writeln;
    end;
    close(output); close(input);
  end.

```

## Задача 5. Домофон

<i>Входной файл</i>	intercom.in
<i>Выходной файл</i>	intercom.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Последнее время на подъездах в Нижнем Новгороде стало популярно ставить домофоны. Домофон представляет собой цифровую клавиатуру, на которой посетитель должен набрать номер нужной ему квартиры и нажать клавишу вызова, после чего жители этой квартиры могут открыть ему дверь (а могут и не открыть).

При интенсивном использовании домофона краска, нанесённая на цифровые клавиши, постепенно стирается. Одна из компаний, занимающихся установкой домофонов, заинтересовалась возможностью определить диапазон номеров квартир, расположенных в подъезде, по тому, насколько стёрлась краска с клавиш. Они уже обнаружили, что по состоянию клавиши можно определить, сколько раз эту клавишу нажимали. Теперь вам, программисту этой компании, поручили для начала решить простейший вариант задачи восстановления диапазона квартир по «затёртостям». А именно, вашей программе на вход даются 10 чисел — «затёртости» клавиш  $0, 1, \dots, 9$ , т. е. количество раз, которое нажималась соответствующая клавиша. Считая, что за время использования домофона каждую квартиру набрали ровно один раз, и считая, что номера квартир в подъезде начинаются с 1 (т. е. в подъезде расположены квартиры с номерами  $1, 2, \dots, N$  при некотором  $N$ ), определите диапазон квартир в этом подъезде (т. е., фактически, определите это  $N$ ). Считайте, что посетители никогда не набирают ведущих нулей. Считайте, что  $N$  не может превосходить  $10^9$ .

### Формат входных данных

На первой строке входного файла находятся 10 чисел — затёртости цифр  $0, 1, \dots, 9$ . Все затёртости не превышают  $10^9$ ; гарантируется, что есть хотя бы одна ненулевая затёртость.

### Формат выходных данных

В выходной файл выведите одно число  $N$ . Если подходящего набора диапазона квартир не существует, выведите одно число  $-1$ . Если подходящих  $N$  существует несколько, выведите любое. Гарантируется, что, если искомого  $N$  существует, то оно не превосходит  $10^9$ .

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
1 2 1 1 1 1 1 1 1 1	10
2 4 2 2 2 2 2 2 2 2	-1
1 1 0 0 0 0 0 0 0 0	-1
1 0 0 0 0 0 0 0 0 0	-1
162 273 270 263 263 263 262 189 162	826

*Примечание:* В первом примере в подъезде 10 квартир; при этом при наборе номера каждой квартиры ровно по разу единица будет нажата два раза, а остальные цифры — по разу.

## Решение

Идея решения этой задачи состояла в том, что можно искать верхнюю границу диапазона квартир методом деления пополам, потому что каждая «затёртость» монотонно возрастает (не убывает) при увеличении этой границы. Критерием для сравнения наборов «затёртостей» может служить сумма «затёртостей» каждого из наборов. Для решения необходимо уметь считать суммарные «затёртости» цифр набором чисел  $1, 2, \dots, N$  для любого  $N$ . Для этого представим число  $N$  в виде последовательности его цифр  $N = \overline{a_k a_{k-1} \dots a_0} = a_k \cdot 10^k + a_{k-1} \cdot 10^{k-1} + \dots + a_0$ . Цифра  $a_k$  встречается в наборе  $1, 2, \dots, N$  в качестве первой цифры  $k+1$ -значных чисел  $\overline{a_{k-1} \dots a_0} + 1$  раз. Заметим, что «затёртости» цифр от набора чисел  $1, 2, \dots, a_k \cdot 10^k - 1$  выражаются следующим образом:

- «затёртость» цифр 0 равна  $ka_k \cdot 10^k - \overline{11\dots 1}$  ( $k$  единиц)
- «затёртости» цифр от 1 до  $a_k$  равны  $(ka_k + 10) \cdot 10^k$
- «затёртости» цифр от  $a_k + 1$  до 9 равны  $ka_k \cdot 10^k$

Этот результат легко получается индукцией по  $a$  и  $k$  (и остаётся как лёгкое упражнение для участников). Поэтому массив «затёртостей» цифр набором чисел  $1, 2, \dots, N$  получается из трёх слагаемых: «затёртости» цифры  $a_k$  в качестве первой цифры  $(k+1)$ -значных чисел, «затёртостей» цифр набором чисел  $1, 2, \dots, a_k \cdot 10^k$  и оставшиеся «затёртости» цифр набором чисел  $1, 2, \dots, N_1$ , где  $N_1 = \overline{a_{k-1} \dots a_0}$ . Далее аналогично для нового числа  $N_1$  будем искать «затёртости». Стоит, правда, отметить, что в последнем слагаемом необходимо учитывать ведущие нули, которые остались после «отбрасывания» первой цифры. Это влияет лишь на «затёртость» нуля, заменяя её на  $ka \cdot 10^k - k$ . На  $k$ -том шаге число  $N_k$  будет однозначным и все затёртости будут выражаться первыми двумя слагаемыми.

Поскольку в условии не гарантируется, что набор «затёртостей» корректен, то есть отвечает какому-либо значению верхней границы диапазона квартир  $N$ , то при достижении минимально возможного диапазона  $N$  из одного или двух последовательных чисел (когда дальнейшее деление пополам невозможно) необходимо проверить на равенство наборы «затёртостей» с набором, указанным в условии.

## Другой вариант решения

По данному  $N$  посчитать затертости цифр набором чисел  $1, 2, \dots, N$  можно и по-другому. А именно, во-первых, легко определить, сколько раз каждая цифра встречается как *последняя* цифра (цифра единиц) среди этих чисел. Отбросим теперь у каждого числа его последнюю цифру и рассмотрим получившуюся последовательность. Несложно видеть, что в ней будет идти 10 раз подряд число 1, десять раз подряд число 2 и т. д., до некоторого числа (равного неполному частному от деления  $N$  на 10), которое, возможно, встретится меньше десяти раз. Таким образом, видно, что можно задачу вычисления затертостей для числа  $N$  (т. е. вычисления затертостей набором чисел  $1, \dots, N$ ) свести к задаче вычисления затертостей для числа, примерно в 10 раз меньшего.

Таким образом, получаем следующий алгоритм. Поделим  $N$  на 10 с остатком: пусть  $N = 10k + m$ , где  $k$  целое и  $0 \leq m < 10$ . Затертости от чисел  $10k, \dots, 10k+m$  мы позже посчитаем отдельно; сначала вычислим затертости от чисел  $1, 2, \dots, 10k - 1$ .

Если выписать последние цифры чисел  $1, 2, \dots, 10k-1$ , то получится последовательность  $1, 2, \dots, 9, 0, 1, 2, \dots, 9, 0, \dots, 9$ , в которой цифра ноль встречается  $k - 1$  раз, а все остальные цифры —  $k$ . Если же, наоборот, у каждого из этих чисел откинуть последнюю цифру, то получим последовательность чисел  $1, \dots, 1, 2, \dots, 2, 3, \dots, 3, \dots, k - 1$ , в которой каждое число повторено 10 раз. Соответствующие затертости легко вычисляются: надо вычислить затертости для числа  $k - 1$  и умножить их на 10. Итак, затертости для числа  $10k - 1$  вычисляются так: рекурсивно вычисляем затертости для числа  $k - 1$ , потом каждую умножаем на 10 и прибавляем  $k$ , после чего из затертости нуля вычитаем 1.

Осталось вычислить затертости, создаваемые числами  $10k, 10k + 1, \dots, 10k + m$ . Этих чисел не более 10, поэтому особых сложностей вычисление затертостей не представляет. Можно, например, заметить, что последняя цифра у этих чисел пробегает значения  $0, 1, \dots, m$ , т. е. для каждой из этих цифр надо затертость увеличить на 1. Если же откинуть последнюю цифру, то получится одно и то же число  $m + 1$  раз. Можно пройтись по этому числу и для каждой цифры увеличить соответствующую затертость на  $m$ .

Этот алгоритм позволяет достаточно легко вычислить затертости по данному числу  $N$ . Применяв метод деления пополам, можно найти ответ на задачу. Пример реализации этого метода можно посмотреть в решении `kap_k.pas`, которое можно скачать в архиве решений жюри с сайта <http://olympiads.nnov.ru>.

Заметим еще, что если бы в условии гарантировалось, что ответ всегда существует, то можно было бы решать задачу проще. А именно, можно было заметить, что с увеличением  $N$  строго возрастает сумма затертостей. Поскольку сумма затертостей, равная просто суммарному числу цифр в числах от 1 до  $N$ , вычисляется очень легко, то можно было бы написать более простое решение, которое производит деление пополам именно по сумме затертостей. Подобная программа пройдет все тесты, в которых решение действительно существует и потому наберет некоторые баллы, но для полного решения задачи требовалось уметь определять случаи, когда решения нет, что так просто сделать невозможно.

## Пример правильной программы

```

const inf='intercom.in';
      outf='intercom.out';
      NoAnswer='-1';
      Nmax=1000000000;
type TRub=array[0..9]of longint;
var Rub:TRub;
    n,nl,nr:longint;
    ans:boolean;
    i:integer;
    ls:integer;

procedure DoRub(n:longint; var R:TRub);
{Функция считает затёртости для данного N}
var d,expn,exp1,n1:longint;
    l:integer;
    i:integer;
begin
  fillchar(R,sizeof(R),0);
  d:=n;
  n1:=n;
  l:=0;
  expn:=0;
  exp1:=0;
  while d>9 do begin
    d:=d div 10;
    inc(l);
    if expn=0 then
      expn:=1
    else
      expn:=expn*10;
    inc(exp1,expn);
  end;
  inc(exp1,10*expn);
  while l>0 do begin
{Затёртости для чисел 1 .. d*10^(l)-1, где
expn=10^(l-1), n=d*10^(l)+(n mod 10*expn)}
    for i:=0 to 9 do
      inc(R[i],l*d*expn);
    if exp1<>0 then begin
      dec(R[0],exp1);
      exp1:=0;
    end;
    for i:=0 to d-1 do
      inc(R[i],expn*10);
    {Число затёртостей для первой цифры d
числа n в качестве первой цифры}
    n1:=n1 mod (expn*10);
    inc(R[d],n1+1);
    {Число затёртостей для набора цифр с
00..0(10 раз) до n1}
    d:=n1 div expn;
    expn:=expn div 10;
    dec(l);
  end;
  for i:=0 to d do
    inc(R[i]);
  if n=d then
    dec(R[0]);
end;

function less(n:longint):integer;
{Функция проверяет для данного N,
меньше ли все затёртости,
чем данные в задаче. Если меньше,
то выдаёт -1, если больше,
то выдаёт 1, иначе - 0}
var R:TRub;
    i:integer;
    ls:integer;
    bad:boolean;
begin
  DoRub(n,R);
  bad:=false;
  ls:=0;
  for i:=0 to 9 do
    if (R[i]<Rub[i])and(ls=0) then
      ls:=-1
    else if (R[i]>Rub[i])and(ls=0) then

```

```

ls:=1
else if ((R[i]<Rub[i])and(ls=1))
  or((R[i]>Rub[i])and(ls=-1)) then
begin
  ls:=0;
  bad:=true;
  break;
end;
if (not bad)and(ls=0) then ls:=-1;
less:=ls;
end;

function check(n:longint):boolean;
{Функция проверяет для данного N,
совпадает ли набор зажёртостей
с данным в задаче}
var R:TRub;
  i:integer;
begin
  DoRub(n,R);
  check:=true;
  for i:=0 to 9 do
    if R[i]<>Rub[i] then begin
      check:=false;
      break;
    end;
  end;
end;

begin
  assign(input,inf); reset(input);

```

```

for i:=0 to 9 do
  read(Rub[i]);
close(input);
ans:=true;
nl:=1; nr:=Nmax;
repeat
  n:=(nl+nr)div 2;
  ls:=less(n);
  if ls<0 then
    nl:=n
  else if ls>0 then
    nr:=n
  else begin
    ans:=false;
    break;
  end;
until nr-nl<=1 ;
if check(nl) then
  n:=nl
else if check(nr) then
  n:=nr
else
  ans:=false;
assign(output,outf); rewrite(output);
if ans then
  writeln(n)
else writeln(Noanswer);
close(output);
end.

```

## Задача 6. Кривой горизонт

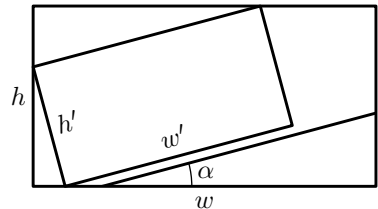
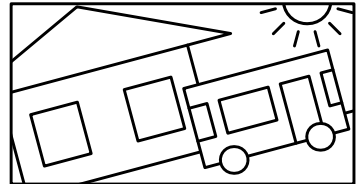
*Входной файл*

*Выходной файл*

*Ограничение по времени*

*Максимальный балл за задачу*

horizon.in  
horizon.out  
1 с  
100



Фирма, в которой работает ваш друг, недавно провела рекламную акцию «Сфотографируй маршрутку — получи приз», в рамках которой пассажиры могли присылать свои фотографии маршруток этой фирмы, а потом специальная конкурсная комиссия выбирала из них лучшие и награждала победителей. Теперь организаторы собираются выставить все фотографии на сайт фирмы.

При подготовке фотографий возникла одна проблема: видимо, некоторые пассажиры фотографировали маршрутки наспех, и потому некоторые фотографии оказались перекошенными: те линии, которые по идее должны быть горизонтальными, оказались наклонными (см. рис.). Администратор сайта категорически отказался выставлять такие фотографии на сайт, порекомендовав организаторам повернуть и обрезать фотографии так, чтобы горизонтальные линии стали на самом деле горизонтальными.

Поэтому организаторы провели большую работу и для каждой фотографии определили, под каким углом на фотографии наклонён горизонт. Теперь им надо вырезать из фотографии прямоугольник, у которого две стороны были бы параллельны горизонту, а две другие, конечно, перпендикулярны. Организаторы хотели вырезать прямоугольник наибольшей возможной площади, удовлетворяющий этому условию, но администратор сайта упростил им работу, потребовав сохранить отношение сторон фотографии. Таким образом, им теперь надо из каждой фотографии вырезать прямоугольный кусок со сторонами, параллельными и перпендикулярными горизонту, и с тем же отношением сторон, что и у оригинальной фотографии, причём среди всех таких прямоугольников выбрав наибольший по площади.

Зная вашего друга не только как хорошего экономиста, но и человека со связями, они обратились к нему с просьбой найти программу, которая помогла бы им выбрать нужный прямоугольник. Ваш друг перенаправил эту просьбу к вам.

Обратите внимание, что порядок сторон администратору сайта важен: если на оригинальной фотографии ширина была  $w$ , а высота  $h$ , а у найденного прямоугольника ширина (т. е. сторона, параллельная горизонту)  $w'$ , а высота (т. е. вторая сторона)  $h'$ , то должно быть  $w : h = w' : h'$ , а не  $w : h = h' : w'$ .

### Формат входных данных

Во входном файле находятся три вещественных числа:  $w$ ,  $h$  и  $\alpha$  — размеры оригинальной фотографии (ширина, высота) и угол наклона горизонта к ширине в градусах (см. рис.). Гарантируется, что  $0 < w \leq 100$ ,  $0 < h \leq 100$ ,  $-90 \leq \alpha \leq 90$ .

Положительные  $\alpha$  обозначают, что горизонт получается из ширины фотографии поворотом против часовой стрелки (как на рисунке), отрицательные  $\alpha$  — по часовой стрелке.

## Формат выходных данных

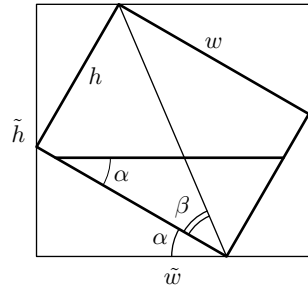
В выходной файл выведите одно число — площадь искомого прямоугольника. Ваш ответ должен отличаться от правильного не более чем на  $10^{-3}$ .

## Пример

Входной файл	Выходной файл
4 3 30	5.1082415465
4 3.0 0	12
4 3 90.0	6.75

## Решение

Эта задача довольно легко решалась из соображений подобия. Повернём нашу фотографию по часовой стрелке на угол  $\alpha$  (так, чтобы горизонт стал действительно горизонтальным) и опишем вокруг неё прямоугольник со сторонами, параллельными сторонам изначальной фотографии. Конкретно, определим габаритные ширину  $\tilde{w}$  и высоту  $\tilde{h}$  у неё (т. е. разность  $x$ -координат самой правой и самой левой точек и разность  $y$ -координат самой верхней и самой нижней; см. рис.). По полученным данным определим, во сколько раз надо сжать этот прямоугольник, чтобы он влезла в изначальную фотографию. А именно, чтобы она влезла по ширине, надо сжать в  $\tilde{w}/w$  раз, чтобы влезла по высоте — в  $\tilde{h}/h$ . Чтобы влезла и по ширине и по высоте, надо сжать в  $k = \max(\tilde{w}/w, \tilde{h}/h)$ , тогда площадь уменьшится в  $k^2$  раз. Тогда ответом на задачу будет  $wh/k^2$ .



Габаритные размеры фотографии после поворота определить достаточно легко. Действительно, очевидно, что искомая ширина будет равна наибольшей из проекций на ось  $x$  диагоналей повернутой фотографии (на самом деле, при  $\alpha > 0$  это всегда будет одна диагональ, а при  $\alpha < 0$  — другая, поэтому, взяв предварительно  $\alpha$  по модулю, можно сразу брать проекцию нужной диагонали). Аналогично, искомая высота равна наибольшей из проекций диагоналей на  $y$ . Эти проекции легко найти, посчитав предварительно угол между диагональю и стороной  $\beta = \arctg(h/w)$ , тогда углы наклона диагоналей к оси  $x$  у повернутой фотографии будут (с точностью до знака)  $\alpha + \beta$  и  $\alpha - \beta$ ; соответственно,

их габаритные размеры будут ширина  $|l \cos(\alpha \pm \beta)|$  и высота  $|l \sin(\alpha \pm \beta)|$ , где  $l = \sqrt{h^2 + w^2}$  — длина диагонали.

На самом деле, можно обойтись без вычисления арктангенса, воспользовавшись формулами пересчёта координат точки при повороте вокруг начала координат на заданный угол или формулами косинуса и синуса суммы и разности, учтя, что  $l \cos \beta = w$  и  $l \sin \beta = h$ .

## Другой вариант решения

Заметим, что абсолютное положение горизонта на фотографии не влияет на ответ, как и знак угла  $\alpha$ , поэтому возьмём  $\alpha$  по модулю. Расположим фотографию так, чтобы две её смежные стороны лежали на осях координат. Предположим, что найдено решение — прямоугольник со сторонами  $kw$  и  $kh$ , где  $0 < k \leq 1$ . Без ограничения общности можно считать, что хотя бы две его вершины лежат на осях координат (этого всегда можно добиться с помощью параллельного переноса).

Сторона, пропорциональная  $w$ , должна стать параллельной горизонту. Тогда две вершины должны иметь координаты  $(0, kw \cos(\alpha))$  и  $(kw \sin(\alpha), 0)$ . Тогда координаты двух других вершин —  $(kh \cos(\alpha), kw \cos(\alpha) + kh \sin(\alpha))$  и  $(kw \sin(\alpha) + kh \cos(\alpha), kh \sin(\alpha))$ . Необходимо, чтобы они лежали внутри прямоугольника, что приводит к условиям, ограничивающим коэффициент пропорциональности  $k$  сверху:

$$kw \cos(\alpha) + kh \sin(\alpha) \leq h \quad \text{и} \quad kw \sin(\alpha) + kh \cos(\alpha) \leq w.$$

Таким образом,

$$k = \min \left( \frac{h}{w \cos(\alpha) + h \sin(\alpha)}, \frac{w}{w \sin(\alpha) + h \cos(\alpha)} \right).$$

Площадь найденного прямоугольника равна  $k^2 wh$ .

## Пример правильной программы

```
{SN+}
var f:text;
    w,h,a:extended;
    b,a1,a2:extended;
    ww,hh:extended;
    t:extended;
    l:extended;

function max(a,b:extended):extended;
begin
    if a>b then max:=a
        else max:=b;
end;

begin
    assign(f,'horizon.in'); reset(f);
    read(f,w,h,a);
    a:=a/180*Pi;
    close(f);
    b:=arctan(h/w);
    l:=sqrt(h*h+w*w);
    a1:=a+b;
```

---

```
a2:=a-b;
ww:=1*max(abs(cos(a1)),abs(cos(a2)));
hh:=1*max(abs(sin(a1)),abs(sin(a2)));
t:=max(ww/w,hh/h);

assign(f,'horizon.out'); rewrite(f);
writeln(f,w*h/t/t:10:10);
close(f);
end.
```

---

## Результаты IV Нижегородской городской олимпиады школьников по информатике

	Школа	Кл.	1	2	3	4	5	6	= Дипл.	
1. P56:	Елифанов Владислав	лиц. 40	10	100	42	100	65	100	100	507 I
2. P13:	Побуринная Оксана	гимн. 2, г. Саров	11	100	46	100	50		100	396 I
3. P31:	Низовцев Сергей	шк. 165	10	50	46	100	55		75	326 I
4. P90:	Крылова Ирина	лиц. 87	10	25		100		66	100	291 I
5. P55:	Люльков Александр	лиц. 40	10	70	29	100	0	14	70	283 I
6. P28:	Тихонов Андрей	шк. 165	10	30	34	100		17	100	281 I
7. P03:	Вадимов Василий	шк. 14, г. Балахна	10	45		100		23	100	268 II
8. P29:	Погорелов Дмитрий	шк. 165	10	5	0	100	0	62	100	267 II
9. P57:	Крот Александр	лиц. 40	10	30	0	100	0	23	65	218 II
10. P07:	Огнева Дарья	лиц. 15, г. Саров	10	30		100		18	65	213 II
11. P58:	Байдасов Марат	лиц. 40	11	45		100		0	65	210 II
12. P01:	Шашкин Юрий	шк. 14, г. Балахна	11	10		100	20	23	50	203 II
13. P60:	Рябкин Николай	лиц. 40	11			100		0	100	200 II
14. P42:	Чистяков Андрей	лиц. 38	10	0	15	83	40	23	25	186 II
15. P59:	Булатов Алексей	лиц. 40	11		1	100	15		65	181 II
16. P49:	Сухов Павел	ЦОД	11	25		100		0	50	175 III
17. P12:	Николайчук Дарья	лиц. 15, г. Саров	8	0	0	100	20	42	10	172 III
18. P26:	Васильев Алексей	шк. 36	11	25	0	100		14	25	164 III
19. P63:	Тюльников Сергей	шк. 121	11	0	9	100			50	159 III
20. P35:	Путилов Андрей	шк. 135	11	50	0	100	0	3		153 III
21. P53:	Зотова Ульяна	лиц. 40	8			100		23	15	138 III
22. P81:	Голованов Антон	шк. 82	10	5		49		18	65	137 III
23. P54:	Фадеев Сергей	лиц. 40	8	0	1	100	0	14	20	135 III
24. P67:	Бовкун Леонид	гимн. 2	11	0	0	100	0	27	0	127 III
24. P34:	Медов Владислав	шк. 36	9	0	0	100	0	27	0	127 III
24. P02:	Петровичев Александр	шк. 14, г. Балахна	11	35	0	49		23	20	127 III
27. P65:	Тюрин Илья	гимн. 2	10	0	0	100	0	0	25	125
27. P04:	Куратова Наталья	лиц. 15, г. Саров	8			100		25		125
29. P33:	Соколов Михаил	шк. 165	9	0	0	100	0	23	0	123

30.	P44: Горячев Никита	лиц. 38	11	20		100				120
30.	P48: Минеева Татьяна	ЦОД	11	20		100				120
32.	P46: Гринес Евгений	шк. 187	11	0	19	100	0	0	0	119
33.	P06: Огнева Мария	гимн. 2, г. Саров	8			100		0	15	115
33.	P11: Григорьева Елена	шк. 3, г. Саров	10	5	0	100		0	10	115
35.	P27: Бычков Илья	шк. 63	11	0	0	83		27		110
36.	P87: Любин Святослав	шк. 183	11			100			5	105
37.	P62: Карпович Евгений	шк. 121	9	0	0	100				100
37.	P45: Нефедов Антон	шк. 186	11			100		0		100
37.	P08: Огнева Ирина	лиц. 15, г. Саров	5	0		100				100
37.	P18: Глазунов Михаил	шк. 149	11			100		0		100
41.	P68: Кочетуров Антон	шк. 180	11		0	83		0	15	98
41.	P91: Борзая Екатерина	лиц. 87	10			83		0	15	98
43.	P30: Вилос Сергей	шк. 165	10	0		83	5	0		88
44.	P82: Анисимова Светлана	шк. 82	9			83		2		85
45.	P39: Сорокоумова Дарья	шк. 48	10		0	77				77
46.	P37: Щекин Илья	шк. 48	11	0	0	49	5		0	54
47.	P93: Кронштадтов Павел	лиц. 87	11						50	50
48.	P84: Сычев Никита	шк. 183	11	0		47				47
48.	P43: Иудин Иван	лиц. 38	10	0	0	47				47
50.	P69: Стукан Артем	шк. 180	9			42				42
51.	P70: Долгополов Евгений	шк. 180	11			41		0		41
52.	P05: Дубатовка Алина	лиц. 15, г. Саров	8	0	0	0		23	0	23
53.	P50: Терёшин Владимир	лиц. 165	8	20	2	0		0		22
54.	P38: Суховерхий Владимир	шк. 32	10	0		0		0	10	10

В таблице не указаны участники, набравшие менее 10 баллов (14 человек).

# V городская олимпиада школьников по информатике 22 января 2009 г.

## Авторы задач

Бударагин Д. И., студент 3 курса ВШОПФ ННГУ;  
Калинин П. А., аспирант ИПФ РАН;  
Круглов А. А., аспирант ИПФ РАН;  
Лелюх В. Д., старший преподаватель ННГУ;  
Побуринная О. Б., студентка 1 курса мехмата МГУ;  
Разенштейн И. П., студент 2 курса мехмата МГУ;  
Тимушев Р. И., студент 6 курса ВШОПФ ННГУ;  
Хаймович И. М., студент 6 курса ВШОПФ ННГУ.

## Задача 1. Счастливые билеты — 3

<i>Входной файл</i>	<code>tickets.in</code>
<i>Выходной файл</i>	<code>tickets.out</code>
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

В фирме, в которой работает ваш друг, ввели новый дизайн билетов на маршрутки. Теперь номер билета может быть любым натуральным числом. Радостные пассажиры тут же придумали новый, очень простой способ определения счастья номера билета. Он состоит в следующем. Пусть номер билета равен  $N$ . Если  $N < 10$ , то счастье числа  $N$  (т. е. и самого билета) равно  $N$ , иначе: посчитаем сумму цифр числа  $N$ , пусть она равна  $S$  — тогда счастье числа  $N$  будет равно счастью числа  $S$ .

Например, счастье билета с номером 7351 равно счастью билета с номером 16 (т. к.  $7 + 3 + 5 + 1 = 16$ ), а она в свою очередь равна счастью билета с номером 7 (т. к.  $1 + 6 = 7$ ), а последняя просто равна 7 (т. к.  $7 < 10$ ).

Такое определение счастья вполне устроило обычных пассажиров маршрутки, но совсем не устроило большинство студентов, которые быстро нашли способ без особенных усилий определять счастье

билета. Поэтому, используя это определение, они придумали новую игру: обладатель билета должен разложить его номер в сумму нескольких чисел так, чтобы суммарная счастливость слагаемых была максимальной.

Но и эта игра устроила не всех студентов. Наиболее активные из них заметили, что игра становится ещё более интересной, если раскладывать  $N$  не в *сумму*, а в *произведение* чисел, с целью, по-прежнему, максимизировать сумму счастливостей множителей.

Напишите программу, которая будет решать эту задачу, т. е. по данному  $N$  находить такое его представление  $N = a_1 \cdot a_2 \cdot \dots$ , где все  $a_i$  натуральны, больше единицы, и суммарная счастливость чисел  $a_1, a_2, \dots$  максимальна.

### Формат входных данных

Для вашего удобства номер билета  $N$  задан его разложением на простые множители. Таким образом, первая строка входного файла содержит одно натуральное число — количество множителей в разложении числа  $N$  на простые, а далее следуют сами множители.  $N$  не превосходит  $10^{18}$ , а каждый простой множитель не превосходит  $10^9$ .

Если оптимальных решений несколько, выведите любое.

### Формат выходных данных

В выходной файл выведите искомое разложение  $N$  на множители. А именно, сначала выведите количество множителей в вашем разложении, а потом — сами множители.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
3	2
2 2 3	6 2
2	2
2 11	2 11

*Примечание:* Если ваша программа будет проходить тесты, в которых  $N \leq 10^9$ , то она получит не менее 30 баллов.

### Решение

Обозначим счастливость числа  $v$  как  $d(v)$ . Эта функция нередко называется цифровым корнем  $v$ .

Для натуральных  $v$  верна следующая формула:

$$d(v) = \begin{cases} v \bmod 9, & v \bmod 9 \neq 0 \\ 9, & v \bmod 9 = 0. \end{cases}$$

Её легко доказать, если заметить, что сумма цифр даёт такой же остаток от деления на 9, как и само число.

Будем работать с простыми множителями, заданными во входном файле. Понятно, что сами множители нас будут интересовать только при выводе ответа, а в процессе вычислений можно заменить все простые множители на их цифровые корни (в силу того, что  $d(a \cdot b) = d(d(a) \cdot d(b))$ , что легко проверяется), и потому мы будем считать, что все множители — натуральные числа от 1 до 9.

Тогда задача решается жадным алгоритмом:

- Все множители, кроме двоек, троек и четвёрок, напрямую выводить в ответ.
- Пока количество троек не меньше двух, удалять две тройки и выдавать 9.
- Пока остались двойки и четвёрки, удалять двойку и четвёрку и выдавать 8.
- Если закончились двойки, то выдать всё оставшееся.
- Теперь считаем, что остались только двойки и не более одной тройки. Пока количество двоек не меньше трёх, удаляем три двойки и выдаём 8.
- Если остались тройка и двойка, то удаляем их и выдаём 6.
- Выдаём всё, что осталось.

Доказательство правильности алгоритма несложное, но слишком перегружено деталями, чтобы приводить его здесь.

## Пример правильной программы

```

{N+}
var f:text;
    n:integer;
    mult:array[1..9,1..60] of comp;
    nmult:array[1..9] of integer;
    ans:array[1..60] of comp;
    nans:integer;
    c:comp;
    dr:integer;
    n_big:comp;
    i,j:integer;

function droot(a:comp):integer;
var r:integer;
begin
    r:=round(a-trunc(a/9)*9);
    if r=0 then r:=9;
    droot:=r;
end;

function pop(i:integer):comp;
begin
    pop:=mult[i,nmult[i]];
    dec(nmult[i]);

```

```

end;
begin
  assign(f,'droot.in');reset(f);
  read(f,n);
  fillchar(nmult,sizeof(nmult),0);
  n_big:=1;
  for i:=1 to n do begin
    read(f,c);
    n_big:=n_big*c;
    dr:=droot(c);
    inc(nmult[dr]);
    mult[dr,nmult[dr]]:=c;
  end;
  close(f);
  nans:=0;
  while nmult[3]>=2 do begin
    inc(nans);
    ans[nans]:=pop(3)*pop(3);
  end;
  while (nmult[4]>=1)and(nmult[2]>=1) do
  begin
    inc(nans);
    ans[nans]:=pop(2)*pop(4);
  end;
  while nmult[2]>=3 do begin
    inc(nans);
    ans[nans]:=pop(2)*pop(2)*pop(2);
  end;
  if (nmult[2]>=1)and(nmult[3]>=1) then
  begin
    inc(nans);
    ans[nans]:=pop(2)*pop(3);
  end;
  for i:=1 to 9 do
    for j:=1 to nmult[i] do begin
      inc(nans);
      ans[nans]:=mult[i,j];
    end;
  end;
  assign(f,'droot.out');rewrite(f);
  writeln(f,nans);
  for i:=1 to nans do
    write(f,ans[i]:0:0,' ');
  close(f);
end.

```

## Задача 2. Олимпийская система — 2

<i>Входной файл</i>	olymp.in
<i>Выходной файл</i>	olymp.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Большое количество соревнований по разным видам спорта проводится по так называемой «олимпийской системе». В простейшем варианте она состоит в следующем. В турнире участвуют  $N = 2^k$  игроков. Они получают номера от 1 до  $N$ , и в первом круге игрок 1 играет с игроком 2, игрок 3 — с игроком 4 и т. д., игрок  $(N - 1)$  — с игроком  $N$ . Проигравшие в каждой паре выбывают из соревнований, а победители проходят во второй круг. Там победитель первой пары (т. е. игрок 1 или игрок 2) играет с победителем второй пары и т. д. Таким образом, в первом круге участвуют  $N = 2^k$  игроков, во втором круге —  $N/2 = 2^{k-1}$  и т. д., в последнем круге (финале) участвуют два игрока.

Серьёзной проблемой при организации соревнований по олимпийской системе является то, что некоторые сильные игроки могут встретиться между собой уже в одном из первых кругов, в результате чего некоторые сильные участники могут рано выбыть из борьбы, в то время как другие, менее сильные участники могут пройти намного выше, если им повезёт с соперниками. Для борьбы с этим эффектом применяется так называемый «рассев» игроков, когда начальная нумера-

ция участников не является полностью случайной, и вероятность «везения»/«невезения» с соперниками резко уменьшается.

В этой задаче вам нужно будет написать программу, находящую в некотором смысле оптимальный «рассев» игроков. Для этого рассмотрим следующую модель.

Предположим, что участников можно упорядочить по силе так, что более сильный игрок всегда будет выигрывать у более слабого. В таком случае по изначальному «расसेву» игроков дальнейший ход соревнований будет однозначно определён. Тогда в качестве критерия оптимальности «рассева» можно потребовать выполнение следующих условий:

- в финале играют двое сильнейших игроков,
- в полуфиналах играют четверо сильнейших игроков,
- и т. д.: в каждом круге играют только сильнейшие игроки (т. е. если в каком-то круге  $n$  мест, то в этом круге должны участвовать  $n$  сильнейших игроков).

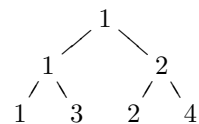
Вам дано  $N$ . Найдите хотя бы один оптимальный рассев.

### Формат входных данных

Входной файл содержит одно число  $N$  — количество участников соревнований. Гарантируется, что  $N$  будет степенью двойки и что  $1 \leq N \leq 2^{17}$ .

### Формат выходных данных

Занумеруем игроков по их силе от 1 до  $N$  (1 — наиболее сильный,  $N$  — наименее). Выведите в выходной файл оптимальный «рассев» участников,



т. е. выведите  $N$  чисел,  $i$ -ое из которых будет номером игрока, который должен стоять на  $i$ -м месте в оптимальном «расसेве».

Если существует несколько оптимальных «рассевов», выведите любой. Если оптимального «рассева» не существует, выведите в выходной файл одно число  $-1$ .

### Пример

Входной файл	Выходной файл
4	1 3 2 4

## Решение

Задача, очевидно, допускает множество решений. По условию задачи число игроков является степенью двойки ( $N = 2^k$ ). При этом несложно показать, что «рассев», помещающий на  $i$ -ое место игрока с номером  $\text{rev}_k(i - 1) + 1$ , будет оптимальным. Здесь  $\text{rev}_k(x)$  — функция, переставляющая биты в двоичном представлении  $k$ -битного числа  $x$  в обратном порядке.

## Пример правильной программы

```

var n, i : longint;
function rev(k,n:longint) : longint;
var r : longint;
begin
  r:=0;
  while n>1 do begin
    r:=(r shl 1)+(k and 1);
    k:=k shr 1;
    n:=n shr 1;
  end; rev:=r;
end;
begin
  assign(input,'olymp.in');
  reset(input);
  assign(output,'olymp.out');
  rewrite(output);
  read(n);
  for i:=0 to n-1 do
    write(rev(i,n)+1,' ');
  close(output);
  close(input);
end.
```

## Задача 3. Автостанции

<i>Входной файл</i>	bus.in
<i>Выходной файл</i>	bus.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Фирма, в которой работает ваш друг, решила воспользоваться удобным моментом и купила компанию, занимающуюся пригородными автобусными пассажирскими перевозками. Таким образом, фирма вашего друга расширяет область деятельности и будет теперь обслуживать и некоторые внутриобластные автобусные маршруты.

Сейчас руководство фирмы, и в том числе ваш друг, заняты оптимизацией работы этих маршрутов. Одна из основных проблем, которые были обнаружены, состоит в том, что большинство автобусов, использующихся там, очень старые и изношенные, и поэтому часто выходят из строя. В целях улучшения ситуации было принято решение о создании сети ремонтных подстанций, которые будут располагаться в некоторых населённых пунктах области и обслуживать другие близлежащие населённые пункты.

Система дорог в области устроена следующим простым образом. Есть  $N$  населённых пунктов, некоторые из которых соединены дорогами. Между каждой парой пунктов существует не более одной дороги,

и более того, для каждой пары населённых пунктов есть ровно один способ добраться из одного в другой (возможно, через промежуточные посёлки).

В каждом населённом пункте можно разместить ремонтную подстанцию. В принципе, фирма может размещать как крупные подстанции, которые даже в одиночку смогут обслуживать всю область, но при этом будут требовать больших расходов на содержание, так и небольшие станции, которые будут обслуживать лишь прилегающие населённые пункты, но при этом будут обходиться намного дешевле. Фирма уже определила, что каждую подстанцию можно характеризовать параметром «мощность», которая может принимать значения, являющиеся целыми положительными числами (равна нулю мощность быть не может). Подстанция с мощностью  $k$  будет обслуживать населённый пункт  $u$ , в котором она расположена, и все другие населённые пункты, до которых можно добраться из  $u$ , используя не более  $k$  дорог (т. е. при  $k = 1$ , например, подстанция обслуживает свой населённый пункт и все, которые напрямую соединены с ним дорогой). Стоимость содержания такой подстанции пропорциональна её мощности.

Теперь перед руководством фирмы и, в частности, вашим другом, стоит задача придумать схему расположения подстанций в населённых пунктах области так, чтобы, во-первых, каждый населённый пункт обслуживался хотя бы одной подстанцией, а во-вторых, суммарная мощность созданных подстанций была минимальна.

Как показывает статистика, автобусы намного реже ломаются на дорогах, чем внутри населённых пунктов, где они вынуждены часто изменять скорость, останавливаться, трогаться с места, заводить двигатель и т. д., поэтому не важно, все ли дороги обслуживаются — главное, чтобы обслуживались все населённые пункты.

### Формат входных данных

В первой строке входного файла находится одно число  $N$  — количество населённых пунктов в области ( $1 \leq N \leq 300$ ). Далее следуют  $N - 1$  строка, описывающая дороги. Каждая строка содержит два числа — номера населённых пунктов, которые соединяет эта дорога. Населённые пункты нумеруются от 1 до  $N$ .

### Формат выходных данных

В первую строку выходного файла выведите одно число — оптимальную суммарную мощность подстанций. Далее выведите  $N$  чисел, описывающих какое-нибудь оптимальное решение.  $i$ -ое из этих чисел долж-

но быть равно мощности подстанции, которую в вашем решении надо расположить в пункте  $i$ , или 0, если в населённом пункте  $i$  не должна находиться подстанция.

### Пример

Входной файл	Выходной файл
5	1
1 2	1 0 0 0 0
1 3	
1 4	
1 5	

*Примечание:* Если ваша программа будет проходить тесты, в которых  $N \leq 30$ , то она получит не менее 30 баллов.

### Решение

Задача решается методом динамического программирования. Для краткости населённые пункты будем называть городами.

Сначала докажем, что существует оптимальное решение, где каждый город обслуживается *ровно одной* станцией. Возьмём любое оптимальное решение. Пусть области действия каких-то двух станций перекрываются. Обозначим города, где они находятся,  $u$  и  $v$ , а их радиусы действия  $R_u$  и  $R_v$  соответственно. Расстояние между  $u$  и  $v$  не превосходит  $R_u + R_v$ , так как области действия перекрываются. Рассмотрим кратчайший путь между  $u$  и  $v$ . На нем найдётся вершина, которая удалена от  $u$  не более, чем на  $R_v$ , а от  $v$  — не более, чем на  $R_u$ . Уберём наши станции и поставим в эту вершину станцию с радиусом действия  $R_u + R_v$ . Несложно заметить, что все города по-прежнему останутся покрытыми, а количество станций уменьшится. Таким образом будем уменьшать количество станций до тех пор, пока будут пересечения областей действия. В конце концов, мы получим оптимальное решение, где области действия станций не пересекаются.

Сначала покажем, как находить искомую минимальную суммарную стоимость, а потом уже научимся восстанавливать расстановку и радиусы станций.

Подвесим дерево за произвольную вершину. Для каждого  $p$  будем пытаться решать нашу задачу для поддерева с корнем в вершине  $p$ . Пока неясно, как выразить ответ для  $p$  через ответы для детей  $p$ . В подобных случаях помогает обобщение задачи, что мы и сделаем.

Будем подсчитывать величину  $W(p, q)$ , где  $p$  — корень поддерева, а  $q$  — целое число. Смысл величины зависит от знака  $q$ :

- Если  $q \geq 0$ , то  $W(p, q)$  — ответ для поддерева с корнем  $p$ , к которому прикрепили цепочку вершин длины  $q$ . При этом должно выполняться дополнительное условие: если прикрепить цепочку длины не  $q$ , а  $q + 1$ , то последняя вершина в цепочке не будет покрыта.
- Если  $q < 0$ , то  $W(p, q)$  — ответ для поддерева с корнем  $p$ , в котором запрещено покрывать вершины, расстояние от которых до  $p$  меньше, чем  $|q|$ .

Проще всего вычислить  $W(p, q)$ , если  $q < 0$ . В этом случае

$$W(p, q) = \sum_{x \in C_p} W(x, q + 1),$$

где  $C_p$  — множество детей  $p$ .

Теперь рассмотрим случай  $q = 0$ . Ставить станцию в  $p$  мы не имеем права. Будем перебирать  $v$  — сына  $p$ , в поддереве которого будет находиться станция, которая покроет  $p$ . Получим следующую формулу:

$$W(p, 0) = \min_{v \in C_p} \left( W(v, 1) + \sum_{\substack{x \in C_p \\ x \neq v}} W(x, 0) \right),$$

где минимум по пустому множеству считается равным  $+\infty$ .

Если  $q > 0$ , то у нас появляется две возможности: ставить в  $p$  станцию радиуса  $q$  или поступать аналогично случаю  $q = 0$ , то есть перебирать сына, из поддерева которого будет покрываться  $p$ . Получим такую формулу:

$$W(p, q) = \min \left\{ \min_{v \in C_p} \left( W(v, q + 1) + \sum_{\substack{x \in C_p \\ x \neq v}} W(x, -q) \right), \right. \\ \left. q + \sum_{x \in C_p} W(x, -q) \right\}.$$

Несложно понять, что ответом будет  $\min_{0 \leq k \leq N} W(r, k)$ , где  $r$  — вершина, за которую мы подвесили дерево.

Таким образом, сложность подсчёта ответа —  $\mathcal{O}(N^2)$ . Восстановить же само решение просто — нужно посмотреть, где достигаются минимумы при подсчёте  $W(p, q)$ , и действовать в соответствии с этим.

Приведённое ниже решение, кроме того, при равных суммарных мощностях минимизирует количество поставленных станций.

## Пример правильной программы

```

type pedge=^tedge;
  tedge=record
    v:integer;
    next:pedge;
  end;
type tiarr=array[-300..300] of integer;
  piarr=^tiarr;
  tbarr=array[-300..300] of byte;
  pbarr=^tbarr;
const inf=1000;
var f:text;
  n:integer;
  gr:array[1..300] of pedge;
  a,b:integer;
  ans:array[1..300] of piarr;
  minn:array[1..300] of pbarr;
  minj:integer;
  gans,gminn:integer;
  mark:array[1..300] of integer;
  i,j:integer;

procedure addedge(a,b:integer);
var p:pedge;
begin
  new(p);
  p^.next:=gr[a];
  p^.v:=b;
  gr[a]:=p;
end;

procedure corr(var ans,minn:integer;
  t,tn:integer);
begin
  if (t<ans)or
    ((t=ans)and
    (tn<minn)) then begin
    ans:=t;
    minn:=tn;
  end;
end;

procedure markans(i,p,j:integer); forward;

procedure workout(i,p,j:integer;
  domark:boolean);
var s,sn,t,tn:longint;
  e:pedge;
  from:integer;
  cans,cminn:longint;
begin
  s:=0; sn:=0;
  e:=gr[i];
  while e<>nil do begin
    if e^.v<>p then begin
      s:=s+ans[e^.v]^[-j];
      sn:=sn+minn[e^.v]^[-j];
    end;
    e:=e^.next;
  end;
  if j>0 then begin
    cans:=s+j;
    cminn:=sn+1;
    from:=0;
  end
  else begin
    cans:=inf;
    cminn:=0;
  end;
  e:=gr[i];
  if j<n then begin
    while e<>nil do begin
      if e^.v<>p then begin
        t:=s-ans[e^.v]^[-j]+ans[e^.v]^[-j+1];
        tn:=sn-minn[e^.v]^[-j]
          +minn[e^.v]^[-j+1];
        if (t<cans)
          or((t=cans)and(tn<cminn)) then
          begin
            cans:=t;
            cminn:=tn;
            from:=e^.v;
          end;
        e:=e^.next;
      end;
    end;
  end;
  if not domark then begin
    if cans>1000 then
      cans:=1000;
    ans[i]^[-j]:=cans;
    minn[i]^[-j]:=cminn;
  end
  else begin
    if from=0 then
      mark[i]:=j;
    e:=gr[i];
  end;
end;

```

```

while e<>nil do begin
  if e^.v<>p then begin
    if e^.v=from then
      markans(e^.v,i,j+1)
    else markans(e^.v,i,-j);
  end;
  e:=e^.next;
end;
end;
end;

procedure calc(i,p:integer);
var j:integer;
    e:pedge;
    sn,tn:longint;
begin
  new(ans[i]); new(minn[i]);
  for j:=-300 to 300 do begin
    ans[i]^j]:=inf;
    minn[i]^j]:=0;
  end;
  e:=gr[i];
  while e<>nil do begin
    if e^.v<>p then
      calc(e^.v,i);
    e:=e^.next;
  end;
  for j:=-n to -1 do begin
    sn:=0; tn:=0;
    e:=gr[i];
    while e<>nil do begin
      if e^.v<>p then begin
        inc(sn,ans[e^.v]^j+1];
        inc(tn,minn[e^.v]^j+1];
      end;
      e:=e^.next;
    end;
    if sn>1000 then
      sn:=1000;
    ans[i]^j]:=sn;
    minn[i]^j]:=tn;
  end;
  for j:=0 to n do begin
    workout(i,p,j,false);
  end;
end;
end;

end;

procedure markans(i,p,j:integer);
var e:pedge;
begin
  if j<0 then begin
    e:=gr[i];
    while e<>nil do begin
      if e^.v<>p then
        markans(e^.v,i,j+1);
      e:=e^.next;
    end;
  end
  else
    workout(i,p,j,true);
end;
end;

begin
  assign(f,'bus.in');reset(f);
  read(f,n);
  fillchar(gr,sizeof(gr),0);
  for i:=1 to n-1 do begin
    read(f,a,b);
    addedge(a,b);
    addedge(b,a);
  end;
  calc(1,0);
  gans:=inf;
  for j:=0 to n do
    if (ans[1]^j]<gans)or((ans[1]^j]=gans)
      and(minn[1]^j]<gminn)) then begin
      gans:=ans[1]^j];
      gminn:=minn[1]^j];
      minj:=j;
    end;
  fillchar(mark,sizeof(mark),0);
  markans(1,0,minj);
  assign(f,'bus.out');rewrite(f);
  write(f,gans);
  writeln(f);
  for i:=1 to n do
    write(f,mark[i], ' ');
  close(f);
end.

```

## Задача 4. Протокол матча

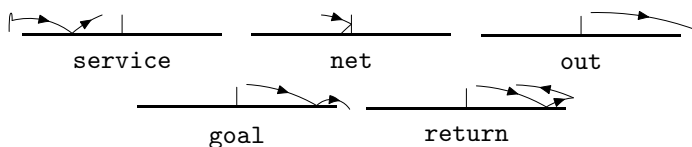
<i>Входной файл</i>	match.in
<i>Выходной файл</i>	match.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

В последнее время в одной из школ Н. Новгорода, а также на одном из факультетов ННГУ стала очень популярна игра в настольный

теннис. Игроки часто сталкиваются со следующей проблемой: довольно трудно уследить за всем ходом матча и при этом не сбиться со счёта, поэтому очень хотелось бы иметь программу, подсчитывающую счёт. Напишите программу, которая по данному протоколу матча восстановит итоговый счёт.

Протокол состоит из последовательности следующих событий: **service**, **net**, **out**, **goal**, **return**, **eam**. События обозначают следующее:

- **service** — подача (при этом игрок ударяет по мячу). **service** — всегда первое событие во входном файле. После него могут следовать **net**, **out**, **goal**, **return**.
- **net** — мяч ударяется о половину поля того игрока, который ударял по мячу последним, слишком много раз. Игрок, который ударял по мячу последним, проигрывает розыгрыш. После этого события могут идти **service** или **eam**.
- **out** — мяч уходит в аут. Игрок, который ударял по мячу последним, проигрывает розыгрыш. После этого события могут идти **service** или **eam**.
- **goal** — игрок, который ударял по мячу последним, забивает гол (т. е. выигрывает розыгрыш). Далее может быть **service** или **eam**.
- **return** — игрок отбивает мяч, ударяя по нему (игроки ударяют по мячу по очереди). Далее может быть **net**, **out**, **goal**, **return**.
- **eam** — матч окончен. Это всегда последнее событие.



Слева на картинках тот, кто последним ударял по шарик, или тот, кто сейчас подаёт

Когда игрок выигрывает розыгрыш, ему начисляется очко. Когда игрок проигрывает розыгрыш, очко начисляется его противнику.

Поддачи подаются по пять штук, т. е. первые пять подач подаёт первый игрок, следующие пять — другой и т. д. Полное количество подач может быть не кратным пяти, в таком случае последняя серия подач будет короче пяти штук.

Конечно, в реальном матче может произойти ситуация, которую невозможно описать этими событиями, но ваша программа должна считать, что весь матч описывается данными во входном файле событиями.

### Формат входных данных

Во входном файле находится список событий. События расположены по одному на строке без пробелов. Последовательность событий удовлетворяет всему, что было сказано выше; пустых строк во входном файле нет (кроме, возможно, строк после события `eom`). Всего событий не более 50 000.

### Формат выходных данных

В выходной файл выведите два числа: очки того, кто подавал первым, потом — очки его противника.

### Пример

<i>Входной файл</i>	<i>Счёт</i>	<i>Выходной файл</i>
service goal service out service net service return return return out service return goal service goal eom	1:0  1:1  1:2      2:2  2:3  2:4	2 4
service out eom	0:1	0 1

## Решение

Эта задача — одна из наиболее простых на олимпиаде. От участника требовалось просто внимательно прочитать условие и написать программу, которая реализовывала бы то, что требуется.

Программу можно было писать, например, следующим образом. Будем нумеровать игроков, начиная с нуля, т. е. нулевой игрок — тот, кто подаёт первым, первый — тот, кто подаёт вторым. В массиве `n` будем хранить текущий счёт, в переменной `cur` будем хранить номер того игрока, который ударял по мячу последним («текущего» игрока). Изначально `n[0]=n[1]=0`; `cur` можно даже не устанавливать, т. к. первое же событие во входном файле `service` установит его (см. ниже).

Далее будем последовательно считывать события из файла и их обрабатывать:

- `service` — надо определить новое значение переменной `cur`, т. е. определить, кто подаёт. Т. к. подаются по пять подач, то номер подающего игрока будет  $(n[0]+n[1]) \text{ div } 5 \text{ mod } 2$ .
- `net` и `out` — для нас совершенно одинаковые события: игрок `cur` проигрывает розыгрыш, поэтому пишем `inc(n[1-cur])`;
- `goal` — наоборот, игрок `cur` выигрывает розыгрыш. `inc(n[cur])`;
- `return` — просто меняется текущий игрок: `cur:=1-cur`;
- `eom` — можно завершать считывание событий.

После команды `eom` завершаем считывание событий и выводим ответ:

```
writeln(f,n[0], ' ',n[1]);
```

## Пример правильной программы

```

const service='service';
      net='net';
      out='out';
      goal='goal';
      return='return';
      eom='eom';
var f:text;
    n:array[0..1] of integer;
    cur:integer;
    s:string;

begin
  assign(f,'match.in');reset(f);
  n[0]:=0;n[1]:=0;
  while true do begin
    readln(f,s);
    if s=service then
      cur:=(n[0]+n[1]) div 5 mod 2
    else if (s=net)or(s=out) then
      inc(n[1-cur])
    else if s=goal then
      inc(n[cur])
    else if s=return then
      cur:=1-cur
    else if s=eom then
      break;
    end;
  close(f);
  assign(f,'match.out');rewrite(f);
  writeln(f,n[0], ' ',n[1]);
  close(f);
end.
```

## Задача 5. Дуга, отрезок и точка

Входной файл

arc.in

Выходной файл

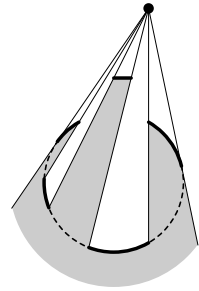
arc.out

Ограничение по времени

1 с

Максимальный балл за задачу

100



На плоскости заданы дуга окружности, отрезок и точка. Как отрезок, так и дуга окружности непрозрачны. Определите, какая часть дуги видна из этой точки.

### Формат входных данных

Входной файл состоит из трёх строк, описывающих данные объекты. Первая строка описывает дугу и содержит пять чисел — координаты центра дуги, радиус дуги, полярный угол точки начала дуги и полярный угол точки конца дуги. Полярные углы заданы в градусах и отсчитываются относительно центра дуги против часовой стрелки от положительного направления оси  $x$ . Вторая строка описывает точку и содержит два числа — её координаты. Третья строка описывает отрезок и содержит четыре числа — координаты начала и конца отрезка. Все числа во входном файле вещественны и не превосходят  $10^6$  по модулю. Гарантируется, что как радиус окружности, так и длина отрезка больше нуля, что полярный угол конца дуги больше, чем полярный угол начала, и что разность этих углов не превосходит  $360$ .

Гарантируются, что никакие два из данных трёх объектов не имеют общих точек.

### Формат выходных данных

Выведите в выходной файл одно число на отрезке от  $0$  до  $1$  — относительную часть дуги, которая видна из данной точки. Ваш ответ должен отличаться от правильного не более, чем на  $10^{-4}$ .

### Пример

Входной файл	Выходной файл
2 1 2 120 420 3 6 2 4 2.5 4	0.496842552858631315

*Примечание:* Пример соответствует рисунку. На рисунке части дуги, которые не видны из точки, нарисованы пунктиром.

## Решение

Задача решается в два этапа.

На первом этапе дуга разбивается на меньшие дуги, каждая из которых либо полностью видна, либо полностью не видна. Для этого необходимо провести касательные к дуге из «глаза», а также прямые, соединяющие «глаз» с концами отрезка и дуги. Точки пересечения этих прямых с дугой и будут концами искомых меньших дуг.

После того, как дуга разбита, необходимо для каждого участка определить, виден ли он. Можно взять какую-нибудь точку внутри участка, например середину, и проверить, видна ли она. Чтобы проверить, видна ли точка, надо проверить, пересекается ли отрезок, который соединяет «глаз» с точкой, с каким-либо объектом.

## Пример правильной программы

```

{${N+}
const eps=1e-3;
var x0,y0:extended;
    x1,y1,x2,y2:extended;
    xc,yc,r:extended;
    beg,en:extended;
    p:array[1..12] of extended;
    np:integer;
    f:text;
    i,j:integer;
    t:extended;
    ans:extended;

function norm(a:extended):extended;
begin
    a:=a-trunc(a/2/Pi)*2*Pi;
    a:=a-8*Pi;
    while a<beg-eps do
        a:=a+2*Pi;
    norm:=a;
end;

procedure addp(a:extended);
begin
    a:=norm(a);
    inc(np);
    p[np]:=a;
end;

function atan2(y,x:extended):extended;
var a:extended;
    r:extended;
begin
    r:=sqrt(x*x+y*y);
    if abs(x)<abs(y) then
        a:=Pi/2-arctan(abs(x/y))
    else a:=arctan(abs(y/x));
    if y<0 then a:=-a;
    if x<0 then a:=Pi-a;
    atan2:=a;
end;

procedure addpoint(x,y:extended);
begin
    addp(atan2(y,x));
end;

procedure addintersect(x1,y1:extended);
var dx,dy:extended;
    d:extended;
    t1,t2:extended;
    a,b,c:extended;
begin
    dx:=x1-x0;
    dy:=y1-y0;
    a:=dx*dx+dy*dy;
    b:=2*(x0*dx+y0*dy);
    c:=x0*x0+y0*y0-r*r;
    d:=b*b-4*a*c;
    if d<eps then exit;
    t1:=(-b+sqrt(d))/2/a;
    t2:=(-b-sqrt(d))/2/a;
    addpoint(x0+dx*t1,y0+dy*t1);
    addpoint(x0+dx*t2,y0+dy*t2);
end;

procedure addtouch;
var a0,da:extended;
    l:extended;
    lt:extended;
begin
    l:=x0*x0+y0*y0;
    if l<r*r+eps then
        exit;

```

```

a0:=atan2(y0,x0);
lt:=sqrt(1-r*r);
da:=arctan(lt/r);
addp(a0+da);
addp(a0-da);
end;

function intersectline(
  x1,y1,x2,y2,
  x3,y3,x4,y4:extended
):boolean;
var d:extended;
    tt,ss:extended;
begin
  intersectline:=false;
  d:=(x2-x1)*(y3-y4)-
    (x3-x4)*(y2-y1);
  if abs(d)<eps then
    exit;
  tt:=(x3-x1)*(y3-y4)-
    (y3-y1)*(x3-x4)/d;
  ss:=(x2-x1)*(y3-y1)-
    (x3-x1)*(y2-y1)/d;
  if (tt<0)or(tt>1) then exit;
  if (ss<0)or(ss>1) then exit;
  intersectline:=true;
end;

function intersectarc(
  x0,y0,x1,y1:extended
):boolean;
var dx,dy:extended;
    d:extended;
    t1,t2:extended;
    a,b,c:extended;
function lies(t:extended):boolean;
var x,y,a:extended;
begin
  lies:=false;
  if (t<eps)or(t>1-eps) then exit;
  x:=x0+dx*t;
  y:=y0+dy*t;
  a:=norm(atan2(y,x));
  lies:=a<en;
end;
begin
  dx:=x1-x0;
  dy:=y1-y0;
  a:=dx*dx+dy*dy;
  b:=2*(x0*dx+y0*dy);
  c:=x0*x0+y0*y0-r*r;
  d:=b*b-4*a*c;
  intersectarc:=false;
  if d<eps then exit;
  t1:=(-b+sqrt(d))/2/a;
  t2:=(-b-sqrt(d))/2/a;
  intersectarc:=lies(t1) or lies(t2);
end;

function check(a:extended):boolean;
var xx,yy:extended;
begin
  xx:=r*cos(a);
  yy:=r*sin(a);
  check:=false;
  if intersectline(x0,y0,xx,yy,
    x1,y1,x2,y2) then
    exit;
  if intersectarc(x0,y0,xx,yy) then
    exit;
  check:=true;
end;

begin
  assign(f,'geometry.in');reset(f);
  read(f,xc,yc,r,beg,en);
  read(f,x0,y0);
  read(f,x1,y1,x2,y2);
  close(f);
  x1:=x1-xc;
  y1:=y1-yc;
  x2:=x2-xc;
  y2:=y2-yc;
  x0:=x0-xc;
  y0:=y0-yc;
  beg:=beg/180*Pi;
  en:=en/180*Pi;
  beg:=beg-trunc(beg/2/Pi)*2*Pi;
  en:=norm(en);
  addp(beg);
  addp(en);
  addintersect(x1,y1);
  addintersect(x2,y2);
  addintersect(r*cos(beg),r*sin(beg));
  addintersect(r*cos(en),r*sin(en));
  addtouch;
  for i:=1 to np do
    for j:=1 to np-1 do
      if p[j]>p[j+1] then begin
        t:=p[j];p[j]:=p[j+1];p[j+1]:=t;
        end;
      ans:=0;
      for i:=1 to np-1 do begin
        if p[i+1]>en then
          break;
        if abs(p[i]-p[i+1])<eps then
          continue;
        if check((p[i]+p[i+1])/2) then
          ans:=ans+p[i+1]-p[i];
        end;
      assign(f,'geometry.out');rewrite(f);
      writeln(f,ans/(en-beg):20:20);
      close(f);
    end.

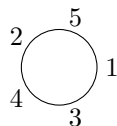
```

## Задача 6. Списывание

Входной файл	table.in
Выходной файл	table.out
Ограничение по времени	1 с
Максимальный балл за задачу	100

На зачёте по квантовым вычислениям преподаватель решил сделать всё возможное, чтобы избежать списывания. За время семестра он успел выделить несколько пар студентов, которые, по его мнению, могут списывать друг у друга, при этом каждый студент входит не более чем в одну такую пару. Теперь преподаватель хочет придумать такую рассадку для студентов, чтобы они все решали задачи самостоятельно.

Зачёт проводится в аудитории, в которой стоит один большой круглый стол, за которым ровно  $N$  мест; студентов в группе тоже ровно  $N$ . Преподаватель считает, что списывать сложнее всего, если тот, у кого списывают, находится на расстоянии ровно  $L$  от того, кто списывает. Таким образом, преподаватель хочет рассадить студентов так, чтобы между каждыми двумя студентами, которые могут списывать друг у друга, сидело бы ровно  $L - 1$  других студентов.



Помогите преподавателю найти такую рассадку.

*Примечание:* Ясно, что в зависимости от того, с какого именно из двух студентов начинать считать, и в зависимости от того, в какую сторону считать, получится разное количество человек, сидящих между ними. Преподавателю требуется лишь, чтобы для каждой пары, начиная с хотя бы какого-нибудь из этих двух студентов, хотя бы в какую-нибудь сторону до другого студента насчитывался бы ровно  $L - 1$  студент.

### Формат входных данных

Первая строка входного файла содержит три числа:  $N$ ,  $L$  и  $K$ , где  $N$  — количество студентов в группе,  $L$  — оптимальное «антисписывательное» расстояние, а  $K$  — количество пар студентов, которые могут друг у друга списывать. Далее следуют  $K$  строк по два числа в каждой — номера студентов, входящих в очередную пару. Студенты нумеруются от 1 до  $N$ .

Все числа во входном файле натуральные и не превосходят 8000; гарантируется, что  $L < N$ .

## Формат выходных данных

Если искомая рассадка существует, то выведите в выходной файл любой из возможных ответов, т. е.  $N$  чисел — номера студентов в порядке обхода стола против часовой стрелки начиная с любого места.

Если решения не существует, выведите  $-1$ .

## Пример

Входной файл	Выходной файл
5 3 2 1 4 2 3	1 5 2 4 3

*Примечание:* Пример соответствует рисунку.

## Решение

Договоримся нумеровать места от 0 до  $N - 1$ . Рассмотрим места с номерами 0,  $L \bmod N$ ,  $2L \bmod N$ ,  $3L \bmod N$  и так далее, то есть просто каждый раз будем откладывать по кругу  $L$  мест, начиная с нуля. Нетрудно видеть, что рано или поздно мы вернёмся в нулевое место, посетив остальные места не более чем по одному разу. Назовём полученное множество посещённых мест *циклом*. Теперь возьмём непосещённое место и проделаем аналогичную операцию. В конце концов, все места разобьются на непересекающиеся циклы.

Каждая пара студентов, подозреваемая в сотрудничестве, должна сидеть в одном цикле, причём на соседних местах. Если у цикла длина  $r$ , то в него можно посадить  $\lfloor \frac{r}{2} \rfloor$  пар студентов (здесь  $\lfloor x \rfloor$  — целая часть числа  $x$ ). Циклы можно заполнять независимо, соответственно, максимальное количество пар, которое можно посадить за стол, равно сумме по всем циклам количества пар, которое можно посадить в этот цикл.

Если это максимальное возможное количество пар меньше  $K$ , то выводим  $-1$ , иначе, пока не кончились пары, сажаем эти пары на соседние места в циклах, а оставшихся студентов рассаживаем на свободные места.

## Пример правильной программы

```

const nmax = 8000;
var ans:array [0..nmax] of integer;
    u,v:array [0..nmax] of boolean;
    N,L,K, x,y, s,t:longint;
    f:boolean;
    i,j:longint;
function next(q:integer):integer;
begin
    next:=(q+L) mod N;
end;

procedure no;
begin
    assign(output,'table.out');

```

```
rewrite(output);
write(-1);
close(input);
close(output);
halt;
end;
begin
  assign(input,'table.in');reset(input);
  read(N,L,K);
  for i:=0 to N do begin
    u[i]:=false;
    v[i]:=false;
    ans[i]:=0;
  end;
  s:=0; t:=0; f:=true;
  for i:=1 to K do begin
    read(x,y);
    v[x]:=true;
    v[y]:=true;
    if ((t=s) and (not f))
      or (next(t)=s) then begin
      u[t]:=true;
```

```
u[next(t)]:=true;
for s:=0 to N do
  if not u[s] then break;
  if s=N then no;
  t:=s;
end;
f:=false;
ans[t]:=x; u[t]:=true; t:=next(t);
ans[t]:=y; u[t]:=true; t:=next(t);
end;
j:=0;
for i:=1 to N do
  if not v[i] then begin
    while ans[j]<>0 do inc(j);
    ans[j]:=i;
    inc(j);
  end;
assign(output,'table.out');rewrite(output);
close(input);
for i:=0 to N-1 do write(ans[i],' ');
close(output);
end.
```

## Результаты V Нижегородской городской олимпиады школьников по информатике

	Школа	Кл.	1	2	3	4	5	6	=	Дипл.
1.	Р59: Люльков Александр	лиц. 40	11	100	100	30	100		66	396 I
2.	Р60: Епифанов Владислав	лиц. 40	11	20	100	50	100	54	66	390 I
3.	Р81: Крылова Ирина	лиц. 87	11	10	100	25	100	13	66	314 I
4.	Р31: Тихонов Андрей	лиц. 165	11	0	93	5	100		100	298 I
5.	Р40: Чистяков Андрей	лиц. 38	11	15	100	10	100	25	45	295 I
6.	Р85: Голованов Антон	шк. 82	11	30	93		100		66	289 I
7.	Р05: Огнева Дарья	лиц. 15, г. Саров	11	5	100	20	100	0	60	285 I
8.	Р32: Погорелов Дмитрий	лиц. 165	11	15	93		100		53	261 I
9.	Р58: Фадеев Сергей	лиц. 40	9	10	93		100		43	246 I
10.	Р35: Терешин Владимир	лиц. 165	9	0	69		100		61	230 II
10.	Р62: Низов Владимир	лиц. 40	11		93		100		37	230 II
12.	Р10: Николайчук Дарья	лиц. 15, г. Саров	9	15	93	10	100	11	0	229 II
13.	Р03: Дубатовка Алина	лиц. 15, г. Саров	9	0	75	5	100		41	221 II
14.	Р30: Низовцев Сергей	лиц. 165	11		100	15	100			215 II
15.	Р61: Крот Александр	лиц. 40	11	5	75	10	100	20	0	210 II
16.	Р66: Серебряков Дмитрий	лиц. 40	11	0	100		100		0	200 II
17.	Р37: Вилов Сергей	лиц. 165	11	25	10		100		52	187 II
18.	Р01: Вадимов Василий	шк. 14, г. Балахна	11	5	100	15	20		46	186 II
19.	Р07: Мосунова Дарья	лиц. 15, г. Саров	11		35		100		42	177 II
20.	Р09: Григорьева Елена	шк. 3, г. Саров	11	0	69	0	100		0	169 II
20.	Р45: Тюрин Илья	гимн. 2	11		69		100		0	169 II
22.	Р82: Борзая Екатерина	лиц. 87	11		20	5	100	16		141 III
23.	Р04: Огнева Мария	гимн. 2, г. Саров	9		0		100		37	137 III
23.	Р11: Побуринная Наталья	гимн. 2, г. Саров	8				100		37	137 III
23.	Р63: Низов Николай	лиц. 40	11		70		25		42	137 III
23.	Р86: Анисимова Светлана	шк. 82	10		69		17		51	137 III
27.	Р87: Кузмичев Дмитрий	шк. 82	7		15	15	100			130 III
28.	Р08: Юкова Юлия	шк. 5, г. Саров	8		15	5	100			120 III
29.	Р36: Макаров Эдгар	лиц. 165	8	15			100			115 III

30.	Р57: Зотова Ульяна	лиц. 40	9	20	10		15	8	61	114	III
31.	Р34: Савиных Александр	лиц. 165	10		15		36		56	107	III
32.	Р18: Василихин Ростислав	лиц. 180	11		100		0			100	III
33.	Р88: Бастраков Илья	шк. 82	11	5	63	0	26	0	0	94	III
34.	Р65: Сорокин Арсений	лиц. 40	6		15		60		16	91	III
35.	Р06: Огнева Ирина	лиц. 15, г. Саров	6	0	75					75	
35.	Р72: Ханова Татьяна	шк. 32	10	0	75					75	
37.	Р89: Меньков Дмитрий	шк. 85	10	0	10		24		0	34	
38.	Р15: Стукан Артем	лиц. 180	10		0		32		0	32	
39.	Р33: Соколов Михаил	лиц. 165	10	5	10	0	14	0	0	29	
40.	Р68: Мельников Игорь	шк. 45	11		10	0	16			26	
41.	Р91: Бушмелев Андрей	шк. 117	11		25					25	
42.	Р14: Коптелов Иван	шк. 148	10	0	10		11			21	
43.	Р51: Карпович Евгений	шк. 121	10		0		20			20	
44.	Р12: Лытов Дмитрий	лиц. 180	7		10		4			14	
45.	Р21: Петрунин Евгений	шк. 149	11	0	10		0			10	
46.	Р44: Фомичев Дмитрий	шк. 24	10		0	0	3		5	8	
47.	Р39: Ложкин Роман	шк. 173	10						7	7	
48.	Р67: Горячкин Артем	шк. 48	11	5		0	0			5	
48.	Р84: Ширяева Екатерина	лиц. 87	7	5						5	

В таблице не указаны участники, набравшие менее 5 баллов (21 человек).



# VI городская олимпиада школьников по информатике 4 февраля 2010 г.

## Авторы задач

Бударагин Д. И., студент 4 курса ВШОПФ ННГУ;  
 Вадимов В. Л., студент 1 курса ВШОПФ ННГУ;  
 Епифанов В. Ю., студент 1 курса мехмата ННГУ;  
 Калинин П. А., аспирант ИПФ РАН;  
 Круглов А. А., младший научный сотрудник ИПФ РАН;  
 Лелюх В. Д., старший преподаватель ННГУ;  
 Люльков А. Е., студент 1 курса ВМК ННГУ;  
 Матросов М. В., студент 4 курса ВМК ННГУ;  
 Низовцев С. Н., студент 1 курса ВМК МГУ;  
 Разенштейн И. П., студент 3 курса мехмата МГУ;  
 Тимухев Р. И., старший разработчик Luxoft/UBS;  
 Тихонов А. О., студент 1 курса ВМК МГУ;  
 Хаймович И. М., аспирант ИФМ РАН;  
 Шмелёв А. С., студент 5 курса ВМК ННГУ.

## Задача 1. Метро

*Входной файл*

*Выходной файл*

*Ограничение по времени*

*Максимальный балл за задачу*

metro.in

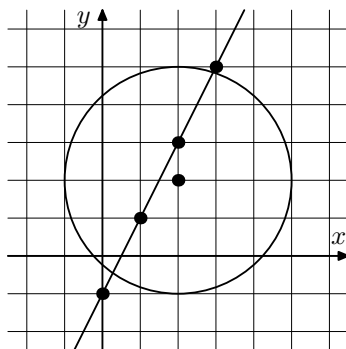
metro.out

1 с

100

В городе  $N$  строят метро. Вася, житель города  $N$ , хочет знать, сколько станций окажутся недалеко от его дома. Помогите ему.

Город  $N$  отличается очень строгой планировкой улиц: каждая улица идёт либо строго с юга на север, либо строго



с востока на запад; при этом расстояние между соседними параллельными улицами одинаково. Соответственно, в городе есть много перекрёстков, расположенных в вершинах квадратной сетки. По плану, первая линия метро будет прямой и будет иметь станции на каждом перекрёстке, через который она пройдёт. Вася считает, что станция находится недалеко от его дома, если расстояние по прямой от его дома до станции не превосходит некоторой фиксированной величины  $R$ .

### Формат входных данных

Введём систему координат с осью  $x$ , направленной с востока на запад, и осью  $y$ , направленной с юга на север, с началом координат на одном из перекрёстков и с единицей длины, равной расстоянию между соседними параллельными улицами. Таким образом, улицы будут прямыми с уравнениями  $\dots, x = -2, x = -1, x = 0, x = 1, x = 2, \dots$ , а также  $\dots, y = -2, y = -1, y = 0, y = 1, y = 2, \dots$ .

Во первой строке входного файла находятся целые числа  $x_0, y_0$  — координаты Васиного дома (считаем, что он находится на некотором перекрёстке), — и расстояние  $R$  в тех же единицах измерения, в которых введены координаты. Во второй строке находятся четыре числа  $x_1, y_1, x_2, y_2$  — координаты некоторых двух различных перекрёстков, через которые пройдёт линия метро. Все координаты во входном файле не превосходят 100 000 000 по модулю; расстояние  $R$  целое, положительное и не превосходит 100 000 000.

Можете считать, что линия метро будет бесконечной в обоих направлениях.

### Формат выходных данных

В выходной файл выведите одно число — количество станций, расположенных недалеко от Васиного дома.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
2 2 3 0 -1 1 1	2
0 0 1 -5 0 -3 0	3

*Примечание:* Первый пример соответствует рисунку; на рисунке дом Васи и станции метро обозначены жирными точками.

## Решение

Фактически, в задаче требовалось по данным прямой и кругу определить, сколько на этой прямой есть точек с целочисленными координатами, лежащих внутри этого круга. Задача достаточно легко решалась следующим образом.

Нам известны две точки («станции») с целочисленными координатами  $(P(x_1, y_1)$  и  $Q(x_2, y_2))$ , лежащие на прямой, но это не обязательно две *ближайшие* станции. Найдём расстояние между *ближайшими* станциями, а точнее, координаты вектора, соединяющего две ближайшие станции. Этот вектор должен иметь минимальные целочисленные координаты и быть пропорционален вектору  $\overrightarrow{PQ}$ ; несложно видеть, что его координаты имеют вид

$$\begin{cases} \Delta x = \frac{x_2 - x_1}{d} \\ \Delta y = \frac{y_2 - y_1}{d}, \end{cases}$$

где  $d = \text{НОД}(x_2 - x_1, y_2 - y_1)$  — наибольший общий делитель координат вектора  $\overrightarrow{PQ}$ .

Теперь несложно видеть, что при любом целом  $t$  формулы

$$\begin{cases} x = x_1 + t \cdot \Delta x \\ y = y_1 + t \cdot \Delta y, \end{cases}$$

определяют координаты некоторой станции; и наоборот, координаты *любой* станции получаются из этих формул при некотором целом  $t$  (в частности, при  $t = 0$  получаем координаты точки  $P$ , а при  $t = d - 1$  — координаты точки  $Q$ ).

Нам требуется определить, сколько из этих точек лежат внутри круга. Точка лежит внутри круга, если для нее выполняется условие

$$(x - x_0)^2 + (y - y_0)^2 \leq R^2.$$

Подставляя сюда выражения для  $x$  и  $y$  и раскрывая скобки, получаем квадратичное неравенство на  $t$  вида

$$at^2 + bt + c \leq 0,$$

где коэффициенты  $a$ ,  $b$  и  $c$  легко выражаются через  $x_1$ ,  $y_1$ ,  $\Delta x$ ,  $\Delta y$ ,  $x_0$ ,  $y_0$  и  $R$ ; в частности, отметим, что всегда  $a = (\Delta x)^2 + (\Delta y)^2 > 0$ .

Это неравенство либо не имеет решений (при  $b^2 - 4ac < 0$ ), либо имеет решения

$$t_1 \leq t \leq t_2,$$

где  $t_1$  и  $t_2$  — корни квадратного уравнения  $at^2 + bt + c = 0$  (с условием  $t_1 \leq t_2$ ; в случае  $b^2 - 4ac = 0$  получим  $t_1 = t_2$ ). С учетом того, что  $t$  должно быть целое, получим

$$\lceil t_1 \rceil \leq t \leq \lfloor t_2 \rfloor,$$

где  $\lceil x \rceil$  — округление  $x$  до ближайшего целого вверх, а  $\lfloor x \rfloor$  — вниз.

Отсюда уже очевидно, что ответ на задачу равен

$$\lfloor t_2 \rfloor - \lceil t_1 \rceil + 1;$$

отметим, что эта формула корректна также и в случае, когда отрезок  $[t_1, t_2]$  не включает ни одного целого числа (и ответ на задачу — ноль), и при  $t_1 = t_2$ .

Таким образом, для решения задачи достаточно реализовать предложенный алгоритм. Отметим также, что особое внимание надо уделить аккуратной работе с вещественными числами с учетом того, что при всех операциях накапливается погрешность; в частности, вместо округления лучше реализовать поиск ближайшей целочисленной точки, лежащей в круге, путем перебора нескольких соседних целочисленных точек и использования только целочисленной арифметики. Еще отметим, что полезно перенести начало координат в точку  $(x_0, y_0)$  (т.е. вычесть  $x_0$  и  $y_0$  из считанных из входного файла координат), что позволит упростить вид коэффициентов  $a$ ,  $b$  и  $c$ .

## Пример правильной программы

```
{$N+}
const eps=1e-5;
var f:text;
    x0,y0,r:longint;
    x1,y1,x2,y2:longint;
    l:longint;
    vx,vy:longint;
    a,b,c:extended;
    d:extended;
    t1,t2:extended;
    n1,n2:longint;

function incircle(t:comp):boolean;
var x,y:comp;
    rr:comp;
begin
    x:=x1+vx*t;
    y:=y1+vy*t;
    rr:=r;
    incircle:=x*x+y*y-rr*rr<=0;
end;

function gcd(a,b:longint):longint;
begin
    if b=0 then gcd:=a
    else gcd:=gcd(b,a mod b);
end;

function ceiltocircle(t:extended):longint;
var r,d:longint;
```

```

begin
  r:=round(t);d:=-5;
  while (not incircle(r+d)and(d<5) do
    inc(d);
    ceiltocircle:=r+d;
end;

function floortocircle(t:extended):longint;
var r,d:longint;
begin
  r:=round(t);d:=5;
  while (not incircle(r+d)and(d>-5) do
    dec(d);
  floortocircle:=r+d;
end;

begin
  assign(f,'metro.in');reset(f);
  read(f,x0,y0,r,x1,y1,x2,y2);
  x1:=x1-x0;
  y1:=y1-y0;
  x2:=x2-x0;
  y2:=y2-y0;
  l:=gcd(x2-x1,y2-y1);
  vx:=(x2-x1) div l;
  vy:=(y2-y1) div l;
  a:=1.0*vx*vx+1.0*vy*vy;
  b:=2.0*x1*vx+2.0*y1*vy;
  c:=1.0*x1*x1+1.0*y1*y1-1.0*R*R;
  d:=b*b-4*a*c;
  if d<-eps then begin
    assign(f,'metro.out');rewrite(f);
    writeln(f,0);
    close(f);
    halt;
  end;
  if d<0 then d:=0;
  t1:=(-b-sqrt(d))/2/a;
  t2:=(-b+sqrt(d))/2/a;
  n1:=ceiltocircle(t1);
  n2:=floortocircle(t2);
  assign(f,'metro.out');rewrite(f);
  if n1<n2 then writeln(f,n2-n1+1)
  else writeln(f,0);close(f);
end.

```

## Задача 2. Автомат со сдачей

<i>Входной файл</i>	change.in
<i>Выходной файл</i>	change.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Фирма, в которой всё ещё работает ваш друг, решила установить в своих маршрутках автоматы по продаже чая и кофе, чтобы во время поездок и, особенно, во время ожидания в пробках, пассажиры могли с толком провести время.

Стоимость стакана чая и кофе в автомате предполагается установить равной пяти рублям. Автоматы будут принимать монеты по 5 и 10 рублей, а также купюры в 10, 50 и 100 рублей. Когда пассажиру надо выдавать сдачу (т.е. когда пассажир бросил в автомат десятирублёвую монету или 10-, 50- или 100-рублёвую купюру), автомат выдаёт сдачу пятирублёвыми монетами; если же пассажир бросил в автомат пятирублёвую монету, то автомат её сохраняет и может использовать для сдачи следующим пассажирам.

Ясно, что, чтобы обеспечить возможность выдачи сдачи всем покупателям, может потребоваться изначально загрузить в автомат некоторое количество пятирублёвых монет. Сейчас на маршрутках фирмы проходят испытания с целью определить минимальное количество монет, которые надо загрузить в автомат перед выездом маршрутки в

рейс. Вам дан протокол одного из таких испытаний: известен порядок, в котором пассажиры оплачивали свои покупки различными монетами и купюрами. Определите, какое минимальное количество пятирублёвых монет должно было изначально находиться в автомате, чтобы всем пассажирам хватило сдачи.

### Формат входных данных

В первой строке входного файла находится одно натуральное число  $N$  — количество покупок в автомате, которые были совершены в ходе испытания ( $1 \leq N \leq 50\,000$ ). Во второй строке находятся  $N$  натуральных чисел, каждое из которых равно номиналу монеты или купюры, которую использовал очередной покупатель для оплаты; каждый номинал может принимать одно из четырёх значений: 5, 10, 50 или 100.

### Формат выходных данных

В выходной файл выведите одно число — минимальное количество пятирублёвых монет, которые надо было загрузить в автомат изначально, чтобы всем покупателям хватило сдачи.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
3 10 5 100	19
3 5 5 10	0
4 50 5 5 5	9

*Примечание:* В первом примере одна пятирублёвая монета потребуется для сдачи первому покупателю и 19 монет — третьему, но при сдаче третьему можно будет использовать ту монету, которую бросит второй покупатель, поэтому изначально в автомате достаточно 19 монет.

Во втором примере сдачу третьему покупателю можно выдать, используя монету первого или второго покупателя, и поэтому не требуется загружать монеты в автомат изначально.

В третьем примере первому же покупателю требуются девять монет сдачи, и все они должны изначально находиться в автомате.

## Решение

Предположим, что в автомате было  $a$  монет до того, как к нему подошёл первый пассажир. После каждого покупателя число монет в автомате меняется в зависимости от того, сколько денег дал пассажир. Обозначим через  $\delta_i$ , на сколько изменилось число пятирублёвых монет в автомате после покупки  $i$ -го пассажира: если он дал 5 рублей, то число монет увеличивается на одну (т. е.  $\delta_i = 1$ ), а если он дал десяти-, пятидесяти- или сторублёвую купюру, то число монет в автомате уменьшается (нужно дать сдачу), и  $\delta_i$  соответственно равно  $-1$ ,  $-9$  и  $-19$ .

Таким образом, число монет в автомате равно  $a + \delta_1 + \dots + \delta_k$  после покупки  $k$ -го пассажира. Чтобы автомат исправно функционировал, необходимо, чтобы такие суммы,  $a + \sum_{i=1}^k \delta_i$ , были неотрицательными,

то есть  $a \geq -\sum_{i=1}^k \delta_i$  для каждого  $1 \leq k \leq N$ . Таким образом, необходимо, чтобы исходное число монет в автомате  $a$  было больше, чем  $\max_k \left( -\sum_{i=1}^k \delta_i \right) = -\min_k \sum_{i=1}^k \delta_i$ .

Легко понять, что это условие также является и достаточным, поэтому минимальное число монет, которое должно находиться в автомате, равно  $-\min_k \sum_{i=1}^k \delta_i$ , и его легко вычислить по мере чтения входных данных (определяя  $\delta_i$  по величине купюры или монеты покупателя, суммируя их и обновляя минимальное значение).

Отметим также, что может потребоваться особо обработать случай, когда полученное таким образом значение  $a$  отрицательно (что соответствует тому, что первые покупатели накидали в автомат столько монет, что их хватит на всю последующую сдачу и ещё останется). В этом случае надо вывести 0; это можно обработать особо, а можно просто изначально инициализировать переменную, в которой мы будем хранить текущий минимум, значением 0, что и сделано в примере программы.

## Другой вариант решения

Будем считать, что у нас есть два отдельных «резервуара» 5-рублевых монет: монеты, брошенных покупателями, и монеты, загруженные в автомат изначально. Будем также считать, что второй резервуар неограничен: мы можем брать оттуда сколько угодно монет. Но при этом при выдаче сдачи очередному покупателю будем в первую очередь расходовать первый резервуар, и только в том случае, если там монет

не осталось, будем расходовать монеты из второго. Соответственно, в первом резервуаре изначально находится ноль монет, он пополняется, когда пассажиры бросают 5-рублевые монеты, и по возможности расходуется, когда надо выдавать сдачу. Второй же резервуар никогда не пополняется, лишь расходуется по необходимости.

Несложно видеть, что при таком подходе ответом на задачу будет общее количество монет, израсходованных из второго резервуара за все время поездки.

Реализовать такой алгоритм достаточно легко, необходимо лишь хранить количества монет в двух резервуарах в двух отдельных переменных. Пример реализации подобного решения можно посмотреть, например, в решениях участников 13, 15, 16, 30, 51 и некоторых других; решения участников можно скачать с сайта <http://olympiads.nnov.ru>.

Отметим также, что можно реализовать вариант этого решения, считав сначала все действия пассажиров в один массив. Проходя по этому массиву в обратном порядке, легко в специальной переменной поддерживать количество монет, которое необходимо иметь в автомате в данный момент, чтобы было возможно сдать сдачу всем пассажирам, которые будут совершать покупки позже (т.е. тем пассажирам, действия которых мы уже обработали). Подобный алгоритм реализован в решениях участников 63 и 68.

Заметим также, что корректности всех приведенных решений этой задачи очевидны, но не столь очевидна эквивалентность этих решений.

### Пример правильной программы

<pre>var current, coin, min, i, n: longint; begin   current:=0; min:=0;   assign(input, 'change.in');   reset(input);   read(n);   for i:=1 to n do begin     read(coin);     case coin of       5: inc(current);       10: dec(current);</pre>	<pre>50: dec(current,9); 100: dec(current,19); end; if current&lt;min then min:=current; end; close(input); assign(output, 'change.out'); rewrite(output); write(-min); close(output); end.</pre>
---	---

## Задача 3. Логарифм

<i>Входной файл</i>	lg.in
<i>Выходной файл</i>	lg.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Рассмотрим два числа  $a$  и  $b$ . По ним можно однозначно определить

такое целое  $k$ , что

$$b^k \leq a < b^{k+1};$$

это  $k$  мы будем называть целой частью логарифма  $a$  по основанию  $b$ .

Напишите программу, которая будет вычислять целую часть логарифма.

### Формат входных данных

В первой строке входного файла записано одно целое число  $a$  ( $1 \leq a \leq 10^{100}$ ) без ведущих нулей. Во второй строке входного файла записано целое число  $b$  ( $2 \leq b \leq 100$ ).

### Формат выходных данных

В выходной файл выведите одно число — целую часть логарифма  $a$  по основанию  $b$  без ведущих нулей.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>
12345678987654321 3	33
8 2	3
2 5	0

### Решение

Будем решать задачу самым примитивным методом — возводить число  $b$  в степени  $1, 2, \dots, k$ , пока наше число  $b^k$  не станет больше  $a$ . Тогда очевидно, что  $k - 1$  и есть ответ на нашу задачу. Но не очевидно то, что наш алгоритм достаточно быстро найдёт решение и уложится в ограничение по времени. Попробуем оценить результат сверху. Для заданного  $b$  однозначно найдётся такое целое число  $x$ , что  $b^{x-1} \leq 10 < b^x$ . Заметим, что  $a$  всегда строго меньше, чем  $10^n$ , где  $n$  — длина числа  $a$  в десятичной записи. Тогда  $a < 10^n < b^{nx}$ . Следовательно, искомое число  $k$  не больше  $nx$ . При наибольшем допустимом значении  $a$ , равном  $10^{100}$ , значение  $n$  максимально и равно 101. Максимальное возможное значение  $x$  достигается при  $b = 2$ , когда  $x = 4$ . Поэтому для данных ограничений ответ всегда меньше, чем 404. Поэтому несложно видеть, что приведённое решение вполне укладывается в ограничение по времени. Для того, чтобы полностью решить задачу, необходимо реализовать

длинную арифметику. В данной задаче нужны всего лишь две операции — умножение «длинного» числа на «короткое» и сравнение двух «длинных» чисел.

## Пример правильной программы

```

const maxp=100;
      maxl=maxp+4;
type tlong=array[1..maxl] of longint;
var f:text;
      a,t:tlong;
      b:integer;
      pow:integer;

procedure readlong(var a:tlong);
var t,i,j:integer;
      ch:char;
begin
  i:=0;
  repeat
    read(f,ch);
    if not (ch in ['0'..'9']) then break;
    inc(i);
    a[i]:=ord(ch)-48;
  until false ;
  for j:=1 to i div 2 do begin
    t:=a[j];
    a[j]:=a[i+1-j];
    a[i+1-j]:=t;
  end;
end;

function lesseq(var a,b:tlong):boolean;
var i:integer;
begin
  for i:=maxl downto 1 do begin
    if a[i]<b[i] then begin
      lesseq:=true; exit;
    end;
    if a[i]>b[i] then begin
      lesseq:=false; exit;
    end;
  end;
end;

end;
end;
lesseq:=true;
end;

procedure mullong(var a:tlong; b:integer);
var i:integer;
begin
  for i:=1 to maxl do
    a[i]:=a[i]*b;
  for i:=1 to maxl-1 do begin
    a[i+1]:=a[i+1]+a[i] div 10;
    a[i]:=a[i] mod 10;
  end;
end;
end;

begin
  assign(f,'lg.in');reset(f);
  readlong(a);
  read(f,b);
  close(f);
  fillchar(t,sizeof(t),0);
  t[1]:=1;
  pow:=0;
  while lesseq(t,a) do begin
    mullong(t,b);
    inc(pow);
  end;
  dec(pow);
  assign(f,'lg.out');rewrite(f);
  writeln(f,pow);
  close(f);
end.

```

## Задача 4. Шарики

*Входной файл*

balls.in

*Выходной файл*

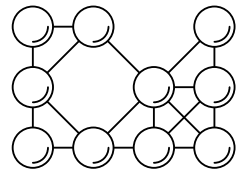
balls.out

*Ограничение по времени*

1 с

*Максимальный балл за задачу*

100



Фирма, в которой работает ваш друг, решила установить на конечной остановке своих маршруток большую абстрактную скульптуру со своим логотипом. Скульптура

будет представлять собой прямоугольную сетку из  $N$  строк и  $M$  столбцов, в некоторых узлах которой будут располагаться разноцветные шары. Для обеспечения жёсткости конструкции шары, расположенные в узлах, соседних по вертикали, горизонтали или диагонали, необходимо соединить металлическими стержнями. Более строго, два шара должны быть соединены стержнем, если разность номеров строк, в которых расположены эти шары, не превосходит по модулю единицы, и разность номеров столбцов тоже не превосходит по модулю единицы.

Напишите программу, которая по заданному расположению шаров позволит определить, как будет выглядеть скульптура после установки стержней.

### Формат входных данных

В первой строке входного файла находятся два натуральных числа  $N$  и  $M$  — размеры конструкции ( $1 \leq N, M \leq 100$ ). Далее следуют  $N$  строк по  $M$  символов в каждой. Каждый символ — это или '#', обозначающий, что в соответствующем узле будет находиться шарик, или '.', обозначающий, что соответствующий узел будет пустой.

### Формат выходных данных

Выведите в выходной файл  $2N - 1$  строку по  $2M - 1$  символов в каждой, изображающие как сами шары, так и соединяющие их стержни. А именно, в нечётных позициях нечётных строк выведите символ '#', или '.', в зависимости от того, заполнен этот узел шариком или нет, а в остальных позициях выведите один из символов ' ' (пробел), '-' (минус), '|' (вертикальная палочка, ASCII #124), '/' (дробь, ASCII #47), '\' (обратный слеш, ASCII #92) или 'X' (латинская заглавная буква X, ASCII #88), отражающий конфигурацию стержней в соответствующем месте структуры.

Если какая-то строка выходного файла должна заканчиваться на пробелы, их можно не выводить.

## Пример

<i>Входной файл</i>	<i>Выходной файл</i>
<pre>3 4 ##.# #.# ####</pre>	<pre>#-# .#  / \ /  # . #-#  \ / X  #-#-#-#</pre>
<pre>3 4 .#.. #.#. .#..</pre>	<pre>. # . . / \ # . # . \ / . # . .</pre>

*Примечание:* Первый пример соответствует рисунку.

## Решение

Задача решалась легко, необходимо было лишь аккуратно реализовать требуемую логику. Можно было сохранить в памяти матрицу размера  $(2N - 1) \times (2M - 1)$  и аккуратно её заполнить, а можно было даже считать строки входного файла по одной и, храня в памяти только текущую и предыдущую строки, выводить данные в выходной файл. Последний вариант и реализован в примере решения.

## Пример правильной программы

```

const maxm=100;
type line=array[1..maxm] of char;
var a1,a2:line;
    n,m,i,j:integer;

procedure draw_line(var l:line);
var i:integer;
begin
  write(l[1]);
  for i:=2 to m do begin
    if (l[i-1]='#') and (l[i]='#') then
      write('-')
    else write(' ');
    write(l[i]);
  end;
  writeln;
end;

procedure draw_between_lines(var l1,l2:line);
var i:integer;
begin
  if (l1[1]='#') and (l2[1]='#') then
    write('|')
    else write(' ');
  for i:=2 to m do begin
    if (l1[i-1]='#') and (l2[i-1]='#') and
      (l1[i]='#') and (l2[i]='#') then
      write('X')
    else
      if (l1[i-1]='#') and (l2[i]='#') then
        write('\')
      else
        if (l2[i-1]='#') and (l1[i]='#') then
          write('/')
        else
          write(' ');
    end;
    if (l1[i]='#') and (l2[i]='#') then
      write('|');
    else
      write(' ');
  end;
  writeln;
end;
begin

```

<pre> assign(input, 'balls.in'); reset(input); assign(output, 'balls.out'); rewrite(output); readln(n,m); for i:=1 to m do read(a2[i]); readln; draw_line(a2); for i:=2 to n do begin </pre>	<pre> a1:=a2; for j:=1 to m do read(a2[j]); readln; draw_between_lines(a1,a2); draw_line(a2); end; close(input); close(output); end. </pre>
--	---

## Задача 5. Совершенно секретно

<i>Входной файл</i>	secret.in
<i>Выходной файл</i>	secret.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

В секретном 240 отделе ИПФ РАН  $N$  сотрудников и  $N$  компьютеров. В отделе вводятся новые требования к секретности. В соответствии с этими требованиями, для каждого сотрудника будут определены ровно  $K$  компьютеров, к которым этот сотрудник будет иметь допуск (т. е. за которыми этот сотрудник будет иметь право работать), причём так, что к каждому компьютеру будут иметь допуск ровно  $K$  сотрудников.

Информация о том, какой сотрудник к какому компьютеру будет иметь допуск, будет известна лишь непосредственно перед вступлением новых требований в силу. Таким образом, чтобы не прерывать работу отдела, сотрудники должны будут быстро решить, кто за каким компьютером будет работать. Для этого им необходимо заранее написать программу, которая по любому распределению допусков сотрудников найдёт рассадку сотрудников по компьютерам, удовлетворяющую этим допускам.

Обратите внимание, что значение  $K$  тоже будет известно лишь в последний момент. Из общих соображений секретности известно лишь, что  $K$  будет равняться или 1, или 2, или 4; поэтому ваша программа должна уметь работать для любого из этих трех значений  $K$ .

### Формат входных данных

В первой строке входного файла записаны натуральные числа  $N$  и  $K$  ( $1 \leq N \leq 500$ ). Далее следуют  $KN$  строк, в каждой из которых записаны два натуральных числа — номер сотрудника и номер компьютера, к которому этот сотрудник имеет допуск.

Гарантируется, что каждый сотрудник имеет допуск ровно к  $K$  компьютерам, что к каждому компьютеру ровно  $K$  сотрудников имеют допуск, и что  $K$  равно либо 1, либо 2, либо 4.

## Формат выходных данных

В выходной файл выведите  $N$  строк, в каждой по два числа — номер сотрудника и номер компьютера, за которым он должен работать. Строки можно выводить в произвольном порядке.

Если есть несколько решений, выведите любое. Можно доказать, что для любого входного файла, удовлетворяющего указанным ограничениям, всегда имеется как минимум одно решение.

## Пример

<i>Входной файл</i>	<i>Выходной файл</i>
3 1 2 3 3 1 1 2	3 1 1 2 2 3
2 2 1 2 2 1 2 2 1 1	1 2 2 1

## Решение

Рассмотрим двудольный граф, в котором одной доле сопоставим работников, а другой — компьютеры. Ребро между вершиной  $i$  из первой доли и вершиной  $j$  из второй будет обозначать, что работник с номером  $i$  имеет допуск к компьютеру с номером  $j$ . Требуется выбрать для каждой вершины из первой доли вершину из второй, причём разным вершинам из первой доли должны соответствовать разные вершины из второй доли.

Случай, когда  $K$  равно 1: каждый работник имеет допуск ровно к одному компьютеру, и только он имеет допуск к этому компьютеру. Всё очевидно, и требуемая рассадка определяется однозначно.

Случай, когда  $K$  равно 2: возьмём любую компоненту связности в этом двудольном графе. Степень каждой вершины в этой компоненте чётна, а значит, в этой компоненте существует эйлеров цикл (цикл, в котором каждое ребро проходится ровно один раз). Удалим из нашего графа каждое второе ребро в этом цикле. После такой операции степень каждой вершины станет равна 1 (ребра из каждой вершины в цикле являются соседними, и одно из них мы оставим, а другое удалим).

Проведя эту операцию с каждой компонентой связности, мы перейдём к уже рассмотренному случаю  $K = 1$ .

Случай, когда  $K$  равно 4, решается аналогично предыдущему, с той лишь разницей, что описанную операцию надо будет произвести дважды: степень каждой вершины после первого удаления станет равной 2, а после второго — 1.

Из деталей реализации отметим необходимость аккуратной работы в случае несвязного графа, а также необходимость корректного удаления ребра. На самом деле, вместо удаления ребер можно на каждой итерации (при каждом уменьшении  $K$  вдвое) формировать новый граф, добавляя в него те ребра, которые необходимо оставить; этот подход и реализован в примере программы. Кроме того, можно не хранить сам эйлеров цикл, а хранить лишь текущую последнюю вершину в цикле и порядковый номер ребра.

Отметим также, что для случая  $K = 2$  существует идейно более простое решение. А именно, несложно показать, что в графе, в котором степень каждой вершины равна 2, каждая компонента связности — это цикл. Действительно, встанем в произвольную вершину и пойдём из неё по какому-нибудь выходящему из неё ребру. Из той вершины, куда мы придём, выходят ровно два ребра, по одному мы пришли — по другому пойдём далее, и т.д. Придя в очередную вершину по одному ребру, пойдём из неё дальше по второму ребру, выходящему из этой вершины. Т.к. граф конечен, мы когда-нибудь вернёмся в вершину, в которой уже были, и несложно видеть, что это будет та вершина, из которой мы начали обход. Таким образом, мы обойдем всю соответствующую компоненту связности, получив цикл, и, если в этом цикле оставить ребра через одно, то получится корректное решение задачи в этой компоненте связности. Повторив это для всех компонент связности, мы решим задачу. Таким образом, можно было решить задачу для  $K = 1$  и 2, не прибегая к поиску эйлеровых циклов; подобное решение набирало 40 баллов.

## Пример правильной программы

```

{$M 65520,0,655360}
const max=1000;
type pedge=~tedge;
      tedge=record
          v:integer;
          next:pedge;
          pair:pedge;
          used:boolean;
end;

```

```

var f:text;
    gr,grnew:array[1..max] of pedge;
    d:array[1..max] of integer;
    dd:integer;
    n,m,k:integer;
    nc:integer;
    i:integer;
    u,v:integer;
    p:pedge;

```

```

was:array[1..max] of integer;
bipart:boolean;
prev:integer;

procedure add(u,v:integer);
var p1,p2:pedge;
begin
  new(p1);
  p1^.v:=v;
  p1^.next:=grnew[u];
  p1^.used:=false;
  grnew[u]:=p1;

  new(p2);
  p2^.v:=u;
  p2^.next:=grnew[v];
  p2^.used:=false;
  grnew[v]:=p2;

  p1^.pair:=p2;
  p2^.pair:=p1;
end;

procedure find(i:integer);
var p:pedge;
q:pedge;
v:integer;
begin
  was[i]:=1;
  while gr[i]<>nil do begin
    if not gr[i]^used then begin
      gr[i]^used:=true;
      gr[i]^pair^.used:=true;
      v:=gr[i]^v;
      gr[i]:=gr[i]^next;
      find(v);
    end
    else gr[i]:=gr[i]^next;
  end;
  inc(nc);
  if nc and 1=0 then
    add(prev,i);
    prev:=i;
  end;
begin
  assign(f,'secret.in');reset(f);
  read(f,n,k);
  m:=k*n;
  n:=n*2;
  fillchar(grnew,sizeof(grnew),0);
  for i:=1 to m do begin
    read(f,u,v);
    v:=v+n div 2;
    add(u,v);
    inc(d[u]);
    inc(d[v]);
  end;
  gr:=grnew;
  dd:=d[1];
  while dd>1 do begin
    nc:=-1;
    prev:=-1;
    fillchar(grnew,sizeof(grnew),0);
    fillchar(was,sizeof(was),0);
    for i:=1 to n do
      if was[i]=0 then begin
        inc(nc);
        find(i);
      end;
    gr:=grnew;
    dd:=dd div 2;
  end;
  assign(f,'secret.out');rewrite(f);
  for i:=1 to n div 2 do begin
    p:=gr[i];
    u:=p^.v;
    writeln(f,i,' ',u-n div 2);
  end;
  close(f);
end.

```

## Задача 6. Шифр

<i>Входной файл</i>	cypher.in
<i>Выходной файл</i>	cypher.out
<i>Ограничение по времени</i>	1 с
<i>Максимальный балл за задачу</i>	100

Жюри N-ской олимпиады по информатике решило зашифровать свои материалы подстановочным шифром. Для шифрования таким шифром задаётся взаимно-однозначное соответствие между буквами алфавита в открытом (т. е. до шифрования) тексте и буквами алфавита

в закрытом (т. е. после шифрования) тексте, это соответствие и является ключом шифра. В процессе шифрования каждая буква в открытом тексте заменяется на соответствующую ей букву в закрытом тексте, порядок букв в слове при этом не меняется.

Однако, память у членов жюри оказалась уже не та, что в молодые годы, поэтому часто они путали, какие буквы надо было заменять на какие. В результате теперь они не могут восстановить свои материалы, а олимпиада уже на носу!

Чтобы разрешить свои проблемы, они обратились к вам. Для облегчения вашей задачи они выписали на бумажку все возможные варианты зашифрования букв, которые они могли применять, в виде набора пар «открытая буква» — «зашифрованная буква». Также вам известны все пары букв  $N$ -ского алфавита, которые могут следовать одна за другой в открытом тексте. Ваша задача состоит в том, чтобы по заданному зашифрованному слову сказать, соответствует ли ему хоть одно расшифрованное слово, единственен ли вариант расшифровки, и привести пример вариантов расшифровки слова. Слово  $A$  считается возможной расшифровкой слова  $B$ , если, во-первых, его можно «зашифровать» (заменяя каждую букву на одну из соответствующих ей «зашифрованных» букв), получив слово  $B$ , и, во-вторых, каждая пара букв слова  $A$ , стоящих рядом, является допустимой для  $N$ -ского языка.

### Формат входных данных

$N$ -ский язык пользуется латинским алфавитом из 26 букв, регистр букв  $N$ -ское жюри не интересуется, поэтому везде в открытом тексте используются большие буквы, а в закрытом — маленькие.

На первой строке входного файла находится одно целое число  $M$  ( $0 \leq M \leq 676$ ) — число пар открытых — «зашифрованных» букв, указанных на бумажке, переданной  $N$ -ским жюри. Далее следуют  $M$  строк, на каждой находятся два символа — сначала открытая буква, потом вариант её «зашифрования». Пары не повторяются.

На следующей строке находится одно целое число  $K$  ( $0 \leq K \leq 676$ ) — число пар открытых букв, которые могут идти одна за другой. Далее следуют  $K$  строк, на каждой из которых по две открытые буквы, образующие такую пару. Пары не повторяются. Заметим, что возможна ситуация, когда последовательность букв ‘AB’ в слове допустима, а ‘BA’ — нет, в этом случае списке будет дана только пара ‘AB’, а пары ‘BA’ не будет.

На следующей строке расположено одно целое число  $N$  ( $1 \leq N \leq 500$ ) — длина зашифрованного слова, а на следующей строке — само

слово ( $N$  маленьких латинских букв).

Может оказаться так, что какой-то открытой букве не соответствует ни одна «зашифрованная»; это означает, что эта буква в открытом тексте не использовалась.

### Формат выходных данных

Если вариантов дешифрования нет ни одного, в первую строку выходного файла выведите 'no'.

Если вариант дешифрования ровно один, в первую строку выходного файла выведите 'only', а во вторую — дешифрованное слово.

Если вариантов дешифрования больше одного, в первую строку выходного файла выведите 'many', а во вторую и третью — любые два различных варианта дешифрования слова.

### Пример

<i>Входной файл</i>	<i>Выходной файл</i>	<i>Входной файл</i>	<i>Выходной файл</i>
1 Uz 0 2 zz	no	7 Aa Az Ax Cz Dv Bx Bz 5 AB CA BC CB DE 4 zzxxx	only BCAB
1 Uz 0 1 z	only U		
2 Uz Xz 3 UU XX XU 2 zz	many UU XU		

### Решение

Задача легко решается методом динамического программирования. Обозначим через  $s$  зашифрованное слово, заданное во входном файле,

пусть  $N$  — его длина. Для каждого числа  $i$  от 1 до  $N$  и для каждой буквы  $\alpha$  от А до Z посчитаем  $W[i, \alpha]$  — число заканчивающихся на букву  $\alpha$  вариантов расшифровки слова  $s[1..i]$  (т. е. подстроки, состоящей из первых  $i$  символов строки  $s$ ). Во-первых, очевидно, что если  $i$ -й символ строки  $s$  ( $s[i]$ ; символы нумеруем с 1) не может расшифровываться в букву  $\alpha$ , то это число равно нулю. Пусть символ  $s[i]$  может расшифровываться в букву  $\alpha$ . Переберём все буквы  $\beta$  и посчитаем количество способов расшифровки строки  $s[1..i]$ , у которых в конце идёт строка  $\beta\alpha$  (т. е. последний символ равен  $\alpha$ , а предпоследний —  $\beta$ ); очевидно, что искомое значение  $W[i, \alpha]$  будет суммой по всем  $\beta$  посчитанных величин. Подсчёт же этого количества при данном  $\beta$  сложностей не вызывает: если буква  $\alpha$  может идти за  $\beta$  в расшифрованном тексте, то ответ равен  $W[i - 1, \beta]$ , иначе равен нулю.

Значения при  $i = 1$  вычисляются очевидным образом: если символ  $s[1]$  может перейти в  $\alpha$ , то  $W[1, \alpha] = 1$ , иначе  $W[1, \alpha] = 0$ . Далее можно в двойном цикле по  $i$  и  $\alpha$  вычислить все значения  $W$ . Таким образом мы сможем получить ответ на первую часть задачи (необходимо только заметить, что в процессе такого решения могут получаться очень большие числа, но, т. к. нас на самом деле интересует, равно ли каждое число нулю, единице, или же оно больше либо равно двум, то можно вместо всех значений, бóльших двух, хранить просто число два).

Вывод решения в задачах на динамическое программирование почти всегда осуществляется довольно просто по насчитанной матрице. Проиллюстрируем это на примере нашей задачи. Пусть для начала нам надо вывести только один любой вариант расшифровки. Для этого можно поступить следующим простым способом. Напишем процедуру  $out(i, \alpha)$ , которая будет выводить в выходной файл произвольную расшифровку строки  $s[1..i]$ , заканчивающихся на символ  $\alpha$  (естественно, при условии, что  $W[i, \alpha] \neq 0$ ). Если  $i = 1$ , то процедура просто выводит символ  $\alpha$  и на этом заканчивает свою работу. Если же  $i \neq 1$ , то процедура, перебравав все буквы, находит такую букву  $\beta$ , что  $W[i - 1, \beta] \neq 0$  и  $\alpha$  может следовать за  $\beta$  (такая буква найдётся, т. к. иначе  $W[i, \alpha]$  было бы равно нулю), вызывает сама себя рекурсивно:  $out(i - 1, \beta)$  и после этого выводит символ  $\alpha$ . Теперь, для вывода какого-нибудь решения основной задачи надо перебрать все буквы, найти такую  $\alpha$ , что  $W[N, \alpha] \neq 0$ , и вызвать  $out(N, \alpha)$ .

Для вывода *двух* решений в случае необходимости можно поступить двумя способами: можно написать процедуру  $out(i, \alpha, w)$ , которая выводит  $w$ -е в каком-нибудь (например, алфавитном) порядке реше-

ние (естественно, при условии  $1 \leq w \leq W[i, \alpha]$ ). Она перебирает все  $\beta$ , «пропуская» решения из  $W[i, \alpha]$  и подсчитывая количество  $w'$  пропущенных решений (в точности как при насчитывании самого значения  $W[i, \alpha]$ , т. е. добавляя количество решений, заканчивающихся на  $\beta\alpha$ , к  $w'$ ). Пусть при некотором  $\beta$  стало  $w' \geq w$ , причём при предыдущем  $\beta$  было  $w' < w$  — обозначим это значение  $w'$  через  $w_0$ . Тогда очевидно, что искомое решение будет соответствовать  $(w - w_0)$ -му среди решений  $W[i - 1, \beta]$ , поэтому вызываем  $out(i - 1, \beta, w - w_0)$ , после чего выводим  $\alpha$ .

Этот способ можно модифицировать, учтя, что нам надо считать только до двух, и поэтому просто можно хранить, пропустили ли мы какое-нибудь решение или ещё пока нет. А именно, пусть мы, перебирая  $\beta$ , наткнулись на  $\beta$ , дающее несколько (т. е. не ноль) решений для наших  $i$  и  $\alpha$ . Тогда: если  $W[i - 1, \beta] \geq 2$ , то вызываем  $out(i - 1, \beta, w)$ , иначе: если мы уже пропускали решения в этом вызове функции *out* **или**  $w = 1$ , то вызываем  $out(i - 1, \beta, 1)$ , иначе пропускаем это (единственное, т. к.  $W[i - 1, \beta] = 1$ ) решение. Этот способ и реализован в примере решения. В таком виде оно не всегда выводит первое и *второе* решения, но всегда первое и какое-то ещё.

Заметим также, что *два* решения можно выводить также и другим способом: а именно, можно выводить первое и *последнее* решения. Это позволит обойтись без сложной логики подсчета решений, описанной в предыдущих двух абзацах, но потребует написания двух отдельных процедур *out*, которые будут различаться порядком перебора букв  $\beta$ : одна будет перебирать их в алфавитном порядке, вторая — в порядке, обратном алфавитному. В итоге (при правильном вызове их из главной программы) первая процедура будет выводить первое в алфавитном порядке решение, а вторая — последнее.

Кроме того, такой подход позволяет также в основной программе считать не до двух, а до одного, т. е. для каждого  $i$  и  $\alpha$  хранить только имеются ли соответствующие решения. Этой информации достаточно для вывода первого и последнего решений, а затем, сравнив их, можно определить, надо выводить *only* или *many*. Эта идея положена в основу решения `kra_j.pas`, которое можно скачать в архиве решений жюри с сайта <http://olympiads.nnov.ru>.

## Пример правильной программы

```
const MaxN=500;
      sss:array[0..2] of string=
          ('no', 'only', 'many');
var f:text;
```

```
can:array['a'..'z', 'A'..'Z'] of
      boolean;
can1:array['A'..'Z', 'A'..'Z'] of
      boolean;
```

```

nn:array[0..MaxN,'A'..'Z']of byte;
s:array[1..MaxN] of char;
ch,ch1,ch2:char;
n:integer;
i:integer;
nall:integer;
n1,n2:integer;

procedure out
  (i:integer;ch:char;n:integer);
var ch1:char;
    was:boolean;
begin
  if i>1 then begin
    was:=false;
    for ch1:='A' to 'Z' do
      if can1[ch1,ch] then
        case nn[i-1,ch1] of
          1: begin
              if was or (n=1) then begin
                out(i-1,ch1,1);
                break;
              end;
              was:=true;
            end;
          2: begin
              out(i-1,ch1,n);
              break;
            end;
        end;
      end;
    end;
    write(f,ch);
  end;

begin
  assign(f,'cypher.in');reset(f);
  readln(f,n);
  fillchar(can,sizeof(can),0);
  for i:=1 to n do begin
    readln(f,ch2,ch1);
    can[ch1,ch2]:=true;
  end;
  readln(f,n);
  fillchar(can1,sizeof(can1),0);
  for i:=1 to n do begin
    readln(f,ch1,ch2);
    can1[ch1,ch2]:=true;
  end;
  readln(f,n);
  for i:=1 to n do begin
    for ch1:='A' to 'Z' do begin
      if nn[n,ch]=2 then begin
        ch1:=ch;ch2:=ch;
        n2:=2;
      end;
      if nn[n,ch]=1 then begin
        if nall=0 then
          ch1:=ch;
        else
          ch2:=ch;
        end;
      inc(nall,nn[n,ch]);
      if nall>=2 then
        break;
      end;
    end;
    if nall>2 then nall:=2;
    writeln(f,sss[nall]);
    if nall>=1 then
      out(n,ch1,n1);
    writeln(f);
    if nall>=2 then
      out(n,ch2,n2);
    writeln(f);
    close(f);
  end.

```

## Результаты VI Нижегородской городской олимпиады школьников по информатике

	Школа	Кл.	1	2	3	4	5	6	= Дипл.	
1. P64:	Калинин Николай	лиц. 40	7	14	100	100	100	20	68	402 I
2. P13:	Николайчук Дарья	лиц. 15, г. Саров	10	35	100	100	100	60		395 I
3. P28:	Терешин Владимир	лиц. 165	10	35	100	100	100	20	38	393 I
4. P10:	Дубатовка Алина	лиц. 15, г. Саров	10	37	100	65	100	50		352 II
5. P27:	Савиных Александр	лиц. 165	11	35	100	90	100	10	13	348 II
6. P14:	Огнева Мария	лиц. 15, г. Саров	10	32	100	100	100	10		342 II
7. P18:	Дробных Клим	14, г. Балахна	11	36	85	95	100	20	0	336 II
8. P61:	Зотова Ульяна	лиц. 40	10	39	100	50	100	40	0	329 II
9. P60:	Баландин Илья	лиц. 40	11	0	100	100	100	10	0	310 II
10. P35:	Вашуров Илья	лиц. 38	10	18	100	75	100	10	0	303 II
11. P15:	Юкова Юлия	5, г. Саров	9	13	100	75	100	10	0	298 II
12. P79:	Кузьмичев Дмитрий	82	8	41	70	70	100	10	0	291 II
13. P63:	Сорокин Арсений	лиц. 40	7		100	65	100	15		280 III
14. P30:	Пауль Эдуард	лиц. 36	9	32	100	75	39	10		256 III
15. P25:	Сokolov Михаил	лиц. 165	11	25	65	30	100	10	0	230 III
16. P16:	Побуринная Наталья	2, г. Саров	9	28	100		100	0		228 III
17. P11:	Огнева Ирина	лиц. 15, г. Саров	7		100		100	25		225 III
18. P37:	Чирков Борис	лиц. 38	11	40	100	55	19	10		224 III
19. P48:	Балагуров Владимир	165	9	40	95	40	35	10		220 III
20. P19:	Земченков Даниил	14, г. Балахна	10	21	100	75	8	10	5	219 III
21. P51:	Лытов Дмитрий	180	8	20	100	40	27	20	0	207 III
22. P69:	Кайнов Олег	30	11	20	90	70	1	20		201 III
23. P32:	Маркин Юрий	лиц. 36	10	32	60	75	8	10		185
24. P33:	Ольшанский Алексей	173	11	28	45	60	23	25		181
25. P56:	Коптелов Иван	148	11	24	50		100			174
26. P81:	Анисимова Светлана	82	11	12	40	100	8	10		170
27. P65:	Кривоносов Михаил	103	9	24	95	50	0			169
28. P76:	Смирнова Марина	лиц. 87	8	0	35	100	23			158
29. P38:	Фомичев Михаил	лиц. 38	11	32	100			10		142

30.	P85: Савченков Вячеслав	183	11	19	60	60	0	0	0	139
31.	P53: Стукан Артем	180	11	24	95		0	15	0	134
32.	P20: Демин Василий	121	10	17	55	60				132
33.	P83: Меньков Дмитрий	85	11	28	55	40	3	5		131
34.	P04: Гринь Илья	73	11	24	30	60		10		124
35.	P86: Егоров Игорь	82	11	18	55	45				118
36.	P68: Митин Антон	лиц. 40	8	10	100			5		115
37.	P54: Исаченко Александр	180	11	24	50	35	1			110
38.	P58: Воронков Арсений	180	10	36	60		6	0		102
39.	P77: Ефремов Максим	лиц. 87	11	21	60		10	10	0	101
40.	P59: Широков Иван	лиц. 40	11		100					100
41.	P21: Карпович Евгений	121	11	37	55					92
42.	P49: Соколов Максим	165	7	17	40	10	6	10	0	83
42.	P62: Дубинов Игорь	лиц. 40	10	10	40	10	23	0	0	83
42.	P82: Носкова Марина	183	11	28	30	15	0	10	0	83
45.	P84: Ушаков Максим	82	10	0	80		1	0		81
46.	P75: Ширяева Екатерина	лиц. 87	8	19	50		0	10		79
47.	P80: Ефимов Евгений	82	9	11	50	10	6	0	0	77
48.	P34: Ложкин Роман	173	11	21	55			0		76
48.	P52: Каримов Динар	180	9	16	45	15				76
50.	P31: Коновалов Дмитрий	лиц. 165	8	24	35	5	6			70
50.	P46: Рыжов Илья	48	11		30	40	0	0		70
52.	P40: Богомолов Дмитрий	лиц. 38	11	24	45	0	0	0		69
53.	P26: Макаров Эдгар	лиц. 165	9		50	0	6	10		66
54.	P06: Ларин Антон	118	10	14	40			10		64
55.	P41: Герасимов Павел	154	11	0	60	0	0	0	0	60
55.	P78: Цверова Юлия	лиц. 87	11	0	60		0	0	0	60
57.	P57: Кожевников Станислав	180	10	0	45					45
58.	P43: Ханова Татьяна	32	11	0	0	35	6	0		41
59.	P08: Гаврилов Дмитрий	149	10		35					35
59.	P47: Чириков Михаил	174	11		30			5		35

В таблице не указаны участники, набравшие 15 баллов и менее (11 человек).

## Список литературы

- [1] Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. *Алгоритмы: построение и анализ*. — М.: Вильямс, 2005.
- [2] Д. Э. Кнут. *Искусство программирования*. — М.: Вильямс, 2006.
- [3] А. Шень. *Программирование: теоремы и задачи*. — М.: МЦНМО, 2004.
- [4] В. И. Беров, А. В. Лапунов, В. А. Матюхин, А. Е. Пономарев. *Особенности национальных задач по информатике*. — Киров: Триада-С, 2000.



Нижегородские городские  
олимпиады школьников по информатике

2005 — 2010

Издание третье, дополненное

Под общей редакцией В. Д. Лелюха

Ответственный за выпуск П. А. Калинин

---

Подписано в печать . . . 2010.  
Формат 60 × 90/16. Бумага офсетная 80 г/м<sup>2</sup>  
Гарнитура ЛН. Печать офсетная.  
Тираж 300 экз. Заказ № 62 (2010).

---

Отпечатано в типографии Института прикладной физики РАН,  
603950, Нижний Новгород, ул. Ульянова, 46