**ВСЕРОССИЙСКАЯ ОЛИМПИАДА ШКОЛЬНИКОВ ПО ИНФОРМАТИКЕ**

# МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ
## по проверке и оцениванию решений задач регионального этапа Всероссийской олимпиады школьников по информатике в 2011/2012 учебном году

Утверждены Центральной предметно-методической комиссией по информатике 28 ноября 2011 г.

**Москва 2011 г.**

## Введение

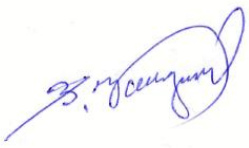
Для проведения регионального этапа Всероссийской олимпиады школьников по информатике тексты олимпиадных заданий, критерии и методики оценки выполненных олимпиадных заданий разрабатывает Центральная предметно-методическая комиссия по информатике (п.п. 16 и 41 Положения о Всероссийской олимпиаде школьников). С учетом этого Центральная предметно-методическая комиссия предоставляет организаторам и жюри регионального этапа 2011/2012 учебного года следующий комплект материалов:

- тексты олимпиадных задач;
- методику проверки решений задач, включая комплекты тестов для каждой задачи в электронном виде;
- проверяющие программы, позволяющие для каждой задачи определять правильность полученного решения в автоматическом режиме;
- описание системы оценивания решений задач;
- методические рекомендации по разбору предложенных олимпиадных задач.

Все вышеназванные материалы являются конфиденциальными и не подлежат преждевременному распространению до завершения регионального этапа по информатике во всех субъектах Российской Федерации.

**Запрещается** также размещать до начала или после завершения регионального этапа все или часть этих материалов на каких-либо региональных или личных сайтах без разрешения Департамента общего образования Минобрнауки России.

Не допускается внесение каких-либо изменений или дополнений в представленные в распоряжение жюри регионального этапа материалы без согласования с Центральной предметно-методической комиссией по информатике. В случае возникновения со стороны регионального жюри каких-либо вопросов или замечаний по условиям задач или системы оценивания необходимо обращаться в Центральную предметно-методическую комиссию по информатике. Для оперативной связи с комиссией можно использовать адрес электронной почты vkiryukhin@nmg.ru .

Председатель Центральной предметно-
методической комиссии по информатике  В.М. Кирюхин

## 1. Комплект олимпиадных задач

Для проведения регионального этапа олимпиады по информатике Центральная предметно-методическая комиссия разработала комплект из восьми задач. Этот комплект задач является единым для всех участников регионального этапа, независимо от класса, в котором они обучаются. Обусловлено это тем фактом, что в большинстве случаев эти задачи являются многоуровневыми, но не с точки зрения формулировки условия задачи, а с точки зрения возможных методов ее решения. В этом случае заданная в условии задачи размерность определяет ее предельную сложность, но при оценке решения участников учитываются не только полные, но и любые частичные (неполные) решения, т.е. решения, полученные для меньшей размерности, чем указано в условии задачи, но реализующие правильно алгоритм решения задачи для этой размерности.

Все задачи пронумерованы от 1 до 8. Задачи с 1-й по 4-у включительно предназначены для проведения первого тура, а задачи с 5-й по 8-у включительно – для проведения второго тура. Номера задач соответствуют их сложности, например, задачи с номерами 1 и 5 являются, по мнению Центральной предметно-методической комиссии, простейшими и должны быть доступны практически всем участникам регионального этапа. В свою очередь, задачи с номерами 4 и 8 являются задачами повышенной сложности и ориентированы на сильнейших участников.

Непосредственно перед началом каждого тура тексты задач для этого тура должны быть растиражированы в количестве экземпляров, соответствующем числу участников олимпиады. На компакт-диске, который также предоставляется организаторам регионального этапа, в папке «Тексты задач для размножения» содержатся готовые для размножения условия задач для каждого тура. Участники должны получить доступ к текстам задач только в момент начала тура.

До окончания тура **категорически запрещается** распространять тексты задач за пределами мест размещения участников олимпиады. Учителя, тренеры, наставники и другие заинтересованные лица могут ознакомиться с содержанием олимпиадных задач тура в местах проведения регионального этапа только после начала тура во всех субъектах Российской Федерации, т.е. после 10.00 по московскому времени в день его проведения.

## Задача 1 «Цапли»

| | |
|---|---|
| Имя входного файла: | `herons.in` |
| Имя выходного файла: | `herons.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 Мбайт |

Петя и Маша пришли в зоопарк. Больше всего Пете понравились цапли. Он был поражен их способностью спать на одной ноге.

В вольере находятся несколько цапель. Некоторые из них стоят на двух ногах, некоторые — на одной. Когда цапля стоит на одной ноге, то другую ее ногу не видно. Петя пересчитал видимые ноги всех цапель, и у него получилось число $a$.

Через несколько минут к вольеру подошла Маша. За это время некоторые цапли могли поменять позу, поэтому Петя предложил ей заново пересчитать видимые ноги цапель. Когда Маша это сделала, у нее получилось число $b$.

Выйдя из зоопарка, Петя с Машей заинтересовались, сколько же всего цапель было в вольере. Вскоре ребята поняли, что однозначно определить это число можно не всегда. Теперь они хотят понять, какое минимальное и какое максимальное количество цапель могло быть в вольере.

Требуется написать программу, которая по заданным числам $a$ и $b$ выведет минимальное и максимальное количество цапель, которое могло быть в вольере.

**Формат входного файла**

Входной файл содержит два целых числа $a$ и $b$, разделенных ровно одним пробелом ($1 \le a \le 10^9$, $1 \le b \le 10^9$).

**Формат выходного файла**

Выведите в выходной файл два целых числа, разделенных пробелом — минимальное и максимальное число цапель, которое могло быть в вольере. Гарантируется, что хотя бы одно количество цапель соответствует условию задачи.

**Пример входных и выходных файлов**

| herons.in | herons.out |
|---|---|
| 3 4 | 2 3 |

**Пояснения к примеру**

В приведенном примере возможны следующие варианты:

1) В вольере две цапли. Когда Петя считал ноги, одна цапля стояла на двух ногах, а другая — на одной. Петя насчитал три ноги. Когда Маша считала ноги, обе цапли стояли на двух ногах, Маша насчитала четыре ноги.

2) В вольере три цапли. Когда Петя считал ноги, все цапли стояли на одной ноге, Петя насчитал три ноги. Когда Маша считала ноги, одна цапля стояла на двух ногах, а еще две — на одной. Маша насчитала четыре ноги.

**Система оценивания**

Правильные решения для тестов, в которых $1 \le a \le 1000$, $1 \le b \le 1000$, будут оцениваться из 50 баллов.

## Задача 2 «Круглый стол»

| | |
|---|---|
| Имя входного файла: | `table.in` |
| Имя выходного файла: | `table.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 Мбайт |

Возрождая древние традиции английских рыцарей, в одном городе члены школьного клуба любителей информатики каждую неделю собираются за круглым столом и обсуждают результаты последних соревнований.

Руководитель клуба Иван Петрович недавно заметил, что не все ребята активно участвуют в обсуждении. Понаблюдав за несколькими заседаниями клуба, он заметил, что активность члена клуба зависит от того, кто с кем сидит рядом.

В клуб приходят на занятия $m$ мальчиков и $n$ девочек. Иван Петрович заметил, что мальчик активно участвует в обсуждении только тогда, когда непосредственно рядом с ним с обеих сторон от него сидят девочки, а девочка активно участвует в обсуждении только тогда, когда непосредственно рядом с ней с одной стороны от нее сидит мальчик, а с другой — девочка.

Желая сделать заседание клуба как можно более интересным, Иван Петрович решил разместить участников за круглым столом таким образом, чтобы как можно больше членов клуба приняло активное участие в обсуждении.

Требуется написать программу, которая по заданным числам $m$ и $n$ выведет такой способ размещения $m$ мальчиков и $n$ девочек за круглым столом, при котором максимальное количество членов клуба будет активно участвовать в обсуждении.

**Формат входного файла**

Входной файл содержит два целых числа $m$ и $n$, разделенных ровно одним пробелом ($0 \le m \le 1000$, $0 \le n \le 1000$, $m + n \ge 3$).

**Формат выходного файла**

Выходной файл должен содержать строку с расположенными в некотором порядке *m* символами «B» (заглавная латинская буква) и *n* символами «G» (заглавная латинская буква). Символ «B» означает мальчика, а символ «G» — девочку.

Символы следует расположить в том порядке, в котором нужно разместить членов клуба вокруг стола. Соседние символы соответствуют членам клуба, которые сидят рядом. Рядом сидят также члены клуба, соответствующие первому и последнему символу выведенной строки.

**Примеры входных и выходных файлов**

| table.in | table.out |
|---|---|
| 1 2 | BGG |
| 2 2 | BGBG |

**Пояснения к примерам**

В первом примере все члены клуба примут активное участие в обсуждении.

Во втором примере мальчики примут активное участие в обсуждении, а девочки нет. В этом примере можно также разместить членов клуба следующим образом: «BBGG». В этом случае активное участие в обсуждении примут обе девочки, а мальчики — нет. Разместить всех так, чтобы три или четыре члена клуба приняли активное участие в обсуждении, нельзя.

## Задача 3 «Поврежденный XML»

| | |
|---|---|
| Имя входного файла: | xml.in |
| Имя выходного файла: | xml.out |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 Мбайт |

Формат XML является распространенным способом обмена данными между различными программами. Недавно программист Иванов написал небольшую программу, которая сохраняет некоторую важную информацию в виде XML-строки.

XML-строка состоит из открывающих и закрывающих тегов.

Открывающий тег начинается с открывающей угловой скобки (<), за ней следует имя тега — непустая строка из строчных букв латинского алфавита, а затем закрывающая угловая скобка (>). Примеры открывающих тегов: <a>, <dog>.

Закрывающий тег начинается с открывающей угловой скобки, за ней следует прямой слеш (/), затем имя тега — непустая строка из строчных букв латинского алфавита, а затем закрывающая угловая скобка. Примеры закрывающихся тегов: </a>, </dog>.

XML-строка называется *корректной*, если она может быть получена по следующим правилам:

- Пустая строка является корректной XML-строкой.
- Если A и B — корректные XML-строки, то строка AB, получающаяся приписыванием строки B в конец строки A, также является корректной XML-строкой.
- Если A — корректная XML-строка, то строка <X>A</X>, получающаяся приписыванием в начало A открывающегося тега, а в конец — закрывающегося с таким же именем, также является корректной XML-строкой. Здесь X — любая непустая строка из строчных букв латинского алфавита.

Например, представленные ниже строки:

```
<a></a>
<a><ab></ab><c></c></a>
<a></a><a></a><a></a>
```

являются корректными XML-строками, а такие строки как:

```
<a></b>
<a><b>
<a><b></a></b>
```

не являются корректными XML-строками.

Иванов отправил файл с сохраненной XML-строкой по электронной почте своему коллеге Петрову. Однако, к сожалению, файл повредился в процессе пересылки: ровно один символ в строке заменился на некоторый другой символ.

Требуется написать программу, которая по строке, которую получил Петров, восстановит исходную XML-строку, которую отправлял Иванов.

**Формат входного файла**

Входной файл содержит одну строку, которая заменой ровно одного символа может быть превращена в корректную XML-строку. Длина строки лежит в пределах от 7 до 1000, включительно. Строка содержит только строчные буквы латинского алфавита и символы «<» (ASCII код 60), «>»(ASCII код 62) и «/»(ASCII код 47).

Строка во входном файле заканчивается переводом строки.

**Формат выходного файла**

Выходной файл должен содержать корректную XML-строку, которая может быть получена из строки во входном файле заменой ровно одного символа на другой. Если вариантов ответа несколько, можно вывести любой.

**Примеры входных и выходных файлов**

| xml.in | xml.out |
|---|---|
| <a></b> | <a></a> |
| <a><aa> | <a></a> |
| <a><>a> | <a></a> |
| <a/</a> | <a></a> |

**Система оценивания**

Правильные решения для тестов, в которых одна буква заменена на другую букву, оцениваются из 50 баллов.

## Задача 4 «Игра с числами»

| | |
|---|---|
| Имя входного файла: | game.in |
| Имя выходного файла: | game.out |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 Мбайт |

Сегодня на уроке математики Петя и Вася изучали понятие арифметической прогрессии. Арифметической прогрессией с разностью $d$ называется последовательность чисел $a_1, a_2, \ldots, a_k$, в которой разность между любыми двумя последовательными числами равна $d$. Например, последовательность 2, 5, 8, 11 является арифметической прогрессией с разностью 3.

После урока Петя и Вася придумали новую игру с числами. Игра проходит следующим образом.

В корзине находятся $n$ фишек, на которых написаны различные целые числа $a_1, a_2, \ldots, a_n$. По ходу игры игроки выкладывают фишки из корзины на стол. Петя и Вася делают ходы по очереди, первым ходит Петя. Ход состоит в том, что игрок берет одну фишку из корзины и выкладывает ее на стол. Игрок может сам решить, какую фишку взять. После этого он должен назвать целое число $d \geq 2$ такое, что все числа на выбранных к данному моменту фишках являются членами некоторой арифметической прогрессии с разностью $d$, не обязательно последовательными. Например, если на столе выложены фишки с числами 2, 8 и 11, то можно назвать число 3, поскольку эти числа являются членами приведенной в начале условия арифметической прогрессии с разностью 3.

Игрок проигрывает, если он не может сделать ход из-за отсутствия фишек в корзине или из-за того, что добавление любой фишки из корзины на стол приводит к тому, что он не сможет подобрать соответствующее число $d$.

Например, если в корзине имеются числа 2, 3, 5 и 7, то Петя может выиграть. Для этого ему необходимо первым ходом выложить на стол число 3. После первого хода у

него много вариантов назвать число $d$, например он может назвать $d = 3$. Теперь у Васи два варианта хода.

1) Вася может вторым ходом выложить фишку с числом 5 и назвать $d = 2$. Тогда Петя выкладывает фишку с числом 7, называя $d = 2$. На столе оказываются фишки с числами 3, 5 и 7, а в корзине осталась только фишка с числом 2. Вася не может ее выложить, поскольку после этого он не сможет назвать корректное число $d$. В этом случае Вася проигрывает.
2) Вася может вторым ходом выложить фишку с числом 7 и также назвать, например, $d = 2$. Тогда Петя выкладывает фишку с числом 5, называя также $d = 2$. Вася снова попадает в ситуацию, когда на столе оказываются фишки с числами 3, 5 и 7, а в корзине осталась только фишка с числом 2, и он также проигрывает.

Заметим, что любой другой первый ход Пети приводит к его проигрышу. Если он выкладывает число 2, то Вася отвечает числом 7, и Петя не может выложить ни одной фишки. Если Петя выкладывает фишку с числом 5 или 7, то Вася выкладывает фишку с числом 2, и у Пети также нет допустимого хода.

Требуется написать программу, которая по заданному количеству фишек $n$ и числам на фишках $a_1, a_2, \ldots, a_n$ определяет, сможет ли Петя выиграть вне зависимости от действий Васи, и находит все возможные первые ходы Пети, ведущие к выигрышу.

**Формат входного файла**

Первая строка входного файла содержит целое число $n$ ($1 \le n \le 200$).

Вторая строка содержит $n$ различных целых чисел $a_1, a_2, \ldots, a_n$ (для всех $i$ от 1 до $n$ выполняется неравенство $1 \le a_i \le 10^5$). Соседние числа разделены ровно одним пробелом.

**Формат выходного файла**

Первая строка выходного файла должна содержать число $k$ — количество различных первых ходов, которые может сделать Петя, чтобы выиграть. Если Вася может выиграть вне зависимости от действий Пети, строка должна содержать цифру 0.

Во второй строке должно содержаться $k$ различных целых чисел — все выигрышные числа. Будем называть число выигрышным, если, выложив в качестве первого хода фишку, содержащую это число, Петя может выиграть вне зависимости от действий Васи. Соседние числа в строке должны быть разделены ровно одним пробелом.

**Примеры входных и выходных файлов**

| game.in | game.out |
|---|---|
| 4<br>2 3 5 7 | 1<br>3 |
| 2<br>2 4 | 0 |

**Пояснения к примерам**

Первый пример рассматривается в тексте условия этой задачи.

Во втором примере, какую бы фишку не выложил Петя первым ходом, Вася в ответ выкладывает другую фишку, и Петя не может сделать ход из-за отсутствия фишек в корзине.

**Система оценивания**

Правильные решения для тестов, в которых $1 \le n \le 15$, будут оцениваться из 40 баллов.

## Задача 5 «Кондиционер»

| | |
|---|---|
| Имя входного файла: | `cond.in` |
| Имя выходного файла: | `cond.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 Мбайт |

В офисе, где работает программист Петр, установили кондиционер нового типа. Этот кондиционер отличается особой простотой в управлении. У кондиционера есть всего лишь два управляемых параметра: желаемая температура и режим работы.

Кондиционер может работать в следующих четырех режимах:

- «freeze» — охлаждение. В этом режиме кондиционер может только уменьшать температуру. Если температура в комнате и так не больше желаемой, то он выключается.
- «heat» — нагрев. В этом режиме кондиционер может только увеличивать температуру. Если температура в комнате и так не меньше желаемой, то он выключается.
- «auto» — автоматический режим. В этом режиме кондиционер может как увеличивать, так и уменьшать температуру в комнате до желаемой.
- «fan» — вентиляция. В этом режиме кондиционер осуществляет только вентиляцию воздуха и не изменяет температуру в комнате.

Кондиционер достаточно мощный, поэтому при настройке на правильный режим работы он за час доводит температуру в комнате до желаемой.

Требуется написать программу, которая по заданной температуре в комнате $t_{room}$, установленным на кондиционере желаемой температуре $t_{cond}$ и режиму работы определяет температуру, которая установится в комнате через час.

**Формат входного файла**

Первая строка входного файла содержит два целых числа $t_{room}$, и $t_{cond}$, разделенных ровно одним пробелом ($-50 \le t_{room} \le 50$, $-50 \le t_{cond} \le 50$).

Вторая строка содержит одно слово, записанное строчными буквами латинского алфавита — режим работы кондиционера, как указано выше.

**Формат выходного файла**

Выходной файл должен содержать одно целое число — температуру, которая установится в комнате через час.

**Примеры входных и выходных файлов**

| cond.in | cond.out |
|---|---|
| 10 20<br>heat | 20 |
| 10 20<br>freeze | 10 |

**Пояснения к примерам**

В первом примере кондиционер находится в режиме нагрева. Через час он нагреет комнату до желаемой температуры в 20 градусов.

Во втором примере кондиционер находится в режиме охлаждения. Поскольку температура в комнате ниже, чем желаемая, кондиционер самостоятельно выключается и температура в комнате не поменяется.

## Задача 6 «Праздничный ужин»

| | |
|---|---|
| Имя входного файла: | dinner.in |
| Имя выходного файла: | dinner.out |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 Мбайт |

Рядом с офисом компании, в которой работает программист Джон, открылось новое кафе. Директор компании решил провести там новогодний ужин.

Меню праздничного новогоднего ужина в кафе состоит из $k$ типов блюд. Для каждого типа блюда есть несколько вариантов на выбор. Всего есть $a_1$ вариантов для первого типа блюда, $a_2$ вариантов для второго типа блюда, и так далее, $a_k$ вариантов для $k$-го типа блюда. Всего, таким образом, предлагается $a_1 \times a_2 \times \ldots \times a_k$ различных заказов праздничного ужина.

Всего на ужине будут присутствовать $m$ сотрудников компании. Каждый сотрудник должен заказать ровно один вариант блюда каждого типа на выбор. Таким образом, ужин каждого сотрудника будет состоять из $k$ блюд. Для того чтобы ужин каждого сотрудника компании был уникален, администратор кафе придумал следующую схему. Сотрудники делают заказ ужина из меню один за другим. Каждый сотрудник выбирает $k$ блюд, по одному варианту каждого типа. После выбора заказа из меню, сотрудник указывает один их типов блюд, и выбранный этим сотрудником вариант блюда этого типа больше не предлагается тем сотрудникам, которые делают заказ после него.

Каждый сотрудник компании запомнил, сколько возможных заказов ужина ему было предложено. Выяснилось, что директору, который выбирал первым, было предложено на выбор $n_1 = a_1 \times a_2 \times \ldots \times a_k$ заказов. Тому, кто выбирал вторым, досталось лишь $n_2 < n_1$ заказов, поскольку один из вариантов одного из типов блюд уже не был доступен, и так далее. Джону, который выбирал последним, был предложен выбор лишь из $n_m$ заказов. Джон заинтересовался, а какое количество вариантов каждого типа блюд было на выбор у директора компании.

Требуется написать программу, которая по заданным числам $k$, $m$ и $n_1$, $n_2$, …, $n_m$ выяснит, какое количество вариантов каждого типа блюд изначально предлагалось на выбор.

**Формат входного файла**

Первая строка входного файла содержит два целых числа $k$ и $m$, разделенных ровно одним пробелом ($1 \le k \le 20$, $2 \le m \le 100$). Вторая строка содержит $m$ чисел: $n_1$, $n_2$, …, $n_m$ (для всех $i$ от 1 до $m$ выполняется неравенство $1 \le n_i \le 10^9$).

**Формат выходного файла**

Выходной файл должен содержать $k$ чисел: $a_1$, $a_2$, …, $a_k$. Если возможных вариантов решения поставленной задачи несколько, требуется вывести любой. Соседние числа должны быть разделены ровно одним пробелом. Гарантируется, что хотя бы одно решение существует.

**Пример входных и выходных файлов**

| dinner.in | Dinner.out |
|---|---|
| 3 3<br>12 8 4 | 3 2 2 |

**Пояснения к примеру**

События в примере могли развиваться, например, следующим образом.

Исходно количество заказов ужина было равно 3×2×2=12. Директор, выбрав свой заказ, указал блюдо первого типа, поэтому второму сотруднику осталось лишь два

варианта блюда первого типа. Количество заказов для него сократилось до 2×2×2 = 8. Он также указал на свое блюдо первого типа, и Джон уже мог выбирать лишь из 1×2×2 = 4 заказов ужина.

## Задача 7 «Космический кегельбан»

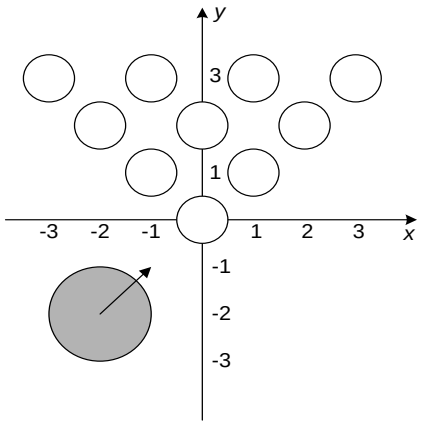
| | |
|---|---|
| Имя входного файла: | `spacepin.in` |
| Имя выходного файла: | `spacepin.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 Мбайт |

На планете Плюк открылся новый космический кегельбан. Поле для кегельбана представляет собой бесконечную плоскость, на которой расставлены кегли.

Каждая кегля представляет собой высокий цилиндр с основанием в виде круга радиусом $r$ метров. Все кегли одинаковые. Кегли расставлены по следующим правилам. Кегли образуют $n$ рядов, в первом ряду стоит одна кегля, во втором — две, и так далее. В последнем $n$-м ряду стоит $n$ кеглей. Введем на плоскости систему координат таким образом, чтобы единица измерения была равна одному километру. Центр единственной кегли в первом ряду находится в точке (0, 0). Центры кеглей во втором ряду находятся в точках (–1, 1) и (1, 1). Таким образом, центры кеглей в $i$-м ряду находятся в точках с координатами $(-(i-1), i-1), (-(i-3), i-1), \ldots, (i-1, i-1)$.

Игра происходит следующим образом. Используется шар с радиусом $q$ метров. Игрок выбирает начальное положение центра шара $(x_c, y_c)$ и вектор направления движения шара $(v_x, v_y)$. После этого шар помещается в начальную точку и двигается, не останавливаясь, в направлении вектора $(v_x, v_y)$. Считается, что шар сбил кеглю, если в процессе движения шара имеет место ситуация, когда у шара и кегли есть общая точка. Сбитые кегли не меняют направления движения шара и не сбивают соседние кегли при падении.

На рисунке приведен пример расположения кеглей для $r = 500$, $n = 4$ и шара для $q = 1000$, $x_c = -2$, $y_c = -2$, $v_x = 1$, $v_y = 1$.



Требуется написать программу, которая по заданным радиусу кегли $r$, количеству рядов кеглей $n$, радиусу шара $q$, его начальному положению $(x_c, y_c)$ и вектору направления движения $(v_x, v_y)$ определяет количество кеглей, сбитых шаром.

**Формат входного файла**

Первая строка входного файла содержит два целых числа: $r$ и $n$, разделенных ровно одним пробелом ($1 \le r \le 700$, $1 \le n \le 200\,000$).

Вторая строка входного файла содержит целое число $q$ ($1 \le q \le 10^9$).

Третья строка входного файла содержит два целых числа $x_c$ и $y_c$, разделенных ровно одним пробелом ($-10^6 \le x_c \le 10^6$, $-10^6 \le y_c$, $1000 \times y_c < -(r + q)$ ).

Четвертая строка входного файла содержит два целых числа $v_x$ и $v_y$, разделенных ровно одним пробелом ($-10^6 \le v_x \le 10^6$, $0 < v_y \le 10^6$).

**Формат выходного файла**

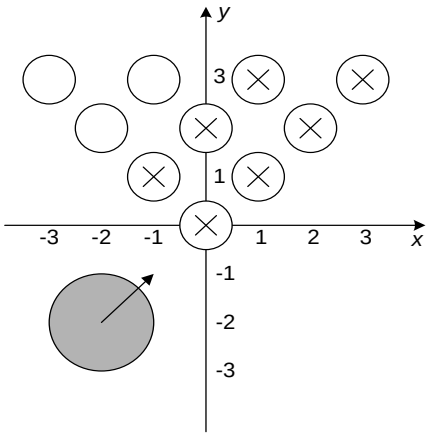
Выходной файл должен содержать одно целое число — количество сбитых кеглей.

**Пример входных и выходных файлов**

| spacepin.in | spacepin.out |
|---|---|
| 500 4<br>1000<br>-2 -2<br>1 1 | 7 |

**Пояснения к примеру**

Рисунок ниже показывает, какие кегли будут сбиты (такие кегли обозначены «x»).




**Система оценивания**

Правильные решения для тестов, в которых $1 \le n \le 1000$ и $v_x = 0$, будут оцениваться из 20 баллов.

Правильные решения для тестов, в которых $1 \le n \le 1000$ и $v_x \ne 0$, будут оцениваться еще из 20 баллов.

Правильные решения для тестов, в которых $1000 < n \le 200\,000$ и $v_x = 0$, будут оцениваться еще из 20 баллов.

Чтобы получить оставшиеся 40 баллов, решение должно правильно работать также для тестов, в которых $1000 < n \le 200\,000$ и $v_x \ne 0$.

## Задача 8 «Abracadabra»

| | |
|---|---|
| Имя входного файла: | `sufpref.in` |
| Имя выходного файла: | `sufpref.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 Мбайт |

Строка $s$ называется *супрефиксом* для строки $t$, если $t$ начинается с $s$ и заканчивается на $s$. Например, «abra» является супрефиксом для строки «abracadabra». В частности, сама строка $t$ является своим супрефиксом. Супрефиксы играют важную роль в различных алгоритмах на строках.

В этой задаче требуется решить обратную задачу о поиске супрефикса, которая заключается в следующем. Задан словарь, содержащий $n$ слов $t_1$, $t_2$, …, $t_n$ и набор из $m$ строк-образцов $s_1$, $s_2$, …, $s_m$. Необходимо для каждой строки-образца из заданного набора найти количество слов в словаре, для которых эта строка-образец является супрефиксом.

Требуется написать программу, которая по заданному числу $n$, $n$ словам словаря $t_1$, $t_2$, …, $t_n$, заданному числу $m$ и $m$ строкам-образцам $s_1$, $s_2$, …, $s_m$ вычислит для каждой строки-образца количество слов из словаря, для которых эта строка-образец является супрефиксом.

**Формат входного файла**

Первая строка входного файла содержит целое число $n$ ($1 \le n \le 200\,000$).

Последующие $n$ строк содержат слова $t_1$, $t_2$, …, $t_n$, по одному слову в каждой строке. Каждое слово состоит из строчных букв латинского алфавита. Длина каждого слова не превышает 50. Суммарная длина всех слов не превышает $10^6$. Словарь не содержит пустых слов.

Затем следует строка, содержащая целое число $m$ ($1 \le m \le 200\,000$).

Последующие $m$ строк содержат строки-образцы $s_1$, $s_2$, …, $s_m$, по одной на каждой строке. Каждая строка-образец состоит из строчных букв латинского алфавита: Длина каждой строки-образца не превышает 50. Суммарная длина всех строк-образцов не превышает $10^6$. Никакая строка-образец не является пустой строкой.

**Формат выходного файла**

Выходной файл должен содержать $m$ чисел, по одному на строке.

Для каждой строки-образца в порядке, в котором они заданы во входном файле, следует вывести количество слов словаря, для которых она является супрефиксом.

**Пример входных и выходных файлов**

| sufpref.in | sufpref.out |
|---|---|
| 4<br>abacaba<br>abracadabra<br>aa<br>abra<br>3<br>a<br>abra<br>abac | 4<br>2<br>0 |

**Система оценивания**

Правильные решения для тестов, в которых $1 \le n \le 100$, $1 \le m \le 100$, будут оцениваться из 30 баллов.

## 2. Методика проверки решений задач

Для проверки решений участников Центральная предметно-методическая комиссия по информатике подготовила для регионального жюри следующие материалы:

- общие методические рекомендации по проверке решений участников;
- комплекты тестов в электронном виде, содержащие для каждой задачи файлы входных данных и соответствующие им файлы выходных данных (для некоторых задач имеются специально разработанные программы – генераторы тестов);
- проверяющие программы, позволяющие для каждой задачи определять правильность полученного решения в автоматическом режиме и программы визуализаторы тестов;
- эталонное решение по каждой задаче для отладки системы автоматической проверки;

Все перечисленные выше материалы для каждой задачи представлены на компакт-диске, который поступает в распоряжение организаторов регионального этапа олимпиады до начала регионального этапа. Материалы для непосредственной проверки решений каждой задачи размещены на этом компакт-диске в отдельной папке с соответствующим именем. Эта папка содержит:

1) Папку «preliminary», содержащую тесты из примеров в условии задачи, предназначенные для предварительной проверки решений участника во время тура. Каждый тест из примера содержится в отдельном файле, входные файлы называются «01», «02», и т.д. Файлы с правильными ответами называются «01.a», «02.a», и т.д. Тесты пронумерованы в том же порядке, в котором они следуют в условии задачи.

2) Папку «tests», содержащую тесты для окончательной проверки и оценивания решений участников, и правильные ответы. Каждый тест содержится в отдельном файле. Входные файлы называются «01», «02» и т.д. Файлы с правильными ответами называются «01.a», «02.a» и т.д.

3) Папку «src», содержащую программы-генераторы, использовавшиеся для создания тестов. Программы-генераторы приведены исключительно для ознакомления жюри регионального этапа с методами получения тестов. Их запуск и использование для перегенерации тестов не требуется, так как комплект материалов уже содержит сгенерированные тесты и ответы к ним. Программы генераторы содержатся в файлах, написанных на различных языках программирования, их компиляция и метод использования описаны в командном файле «buildTests.cmd».

4) Файлы, содержащие проверяющие программы для проверки решений участников с использованием специализированных проверяющих программных систем. Проверяющая программа может содержаться либо в файле «check.dpr», либо в файле «check.cpp». Для компиляции файла «check.dpr» следует использовать компилятор Borland Delphi. Кроме того, при ее исполнении используется библиотека для проверяющих программ testlib, которая содержится в папке «lib» в файле «testlib.pas». Для компиляции файла «check.cpp» можно использовать GNU C++ или Visual C++. Используемая в этом случае библиотека «testlib.h» также содержится в папке «lib».

5) Папку «main», содержащую основное решение жюри. Оно должно использоваться для установки ограничений по времени (см. раздел 2.1, пункт 3).

6) Примеры правильных и некорректных решений. Каждое решение находится в отдельном файле. Этот файл имеет имя, построенное по маске «problem_ab.ext», где «problem» – идентификатор задачи, обычно он совпадает с названием каталога, в котором находятся материалы задачи, «ab» – инициалы автора решения, «ext» – расширение файла, соответствующее языку программирования, на котором написано решение. Решения, которые являются некорректными или неоптимальными, также имеют суффикс, обычно указывающий на проблемы в этом решении, например «problem_ab_wrong.ext» (неправильное решение), «problem_ab_slow.ext» (медленное решение), «problem_ab_bug.ext» (решение, содержащее некоторую типичную ошибку).

Решения предоставляются только для ознакомления членов регионального жюри с возможной реализацией правильных и неправильных решений, а также для проверки работоспособности проверяющих систем. Их использование для генерации правильных ответов на тесты не требуется, так как соответствующие материалы уже содержат готовые правильные ответы на все тесты.

### 2.1. Общие методические рекомендации по проверке решений задач

При проверке решений участников необходимо учитывать следующее.

1. Всем участникам регионального этапа Всероссийской олимпиады школьников, независимо от классов обучения, на каждом туре предлагается один и тот же комплект из четырех задач.

2. Текст каждой задачи содержит описание задачи, максимальное время работы программы на отдельном тесте, размер доступной программе памяти в процессе ее исполнения, форматы входных и выходных данных и примеры входных и выходных данных, варианты оценивания частичных решений.

3. Заданные в условии каждой задачи ограничения на максимальное время работы программы ориентированы на использование при проверке решений участников компьютеров с процессором Core 2 Duo 2.4 ГГц. Именно такого типа компьютеры необходимо использовать для проверки решений участников, причем все решения должны проверяться на одинаковых компьютерах.

Если в распоряжении жюри окажутся компьютеры с другим быстродействием, то рекомендуется установить ограничение по времени в каждой задаче в два раза больше, чем максимальное время работы решения жюри на одном тесте, но не менее 1 секунды. Например, если есть компьютер с процессором Pentium 3 866 МГц и решение жюри работает максимум 2 секунды, то необходимо установить ограничение, равное 4 секундам. В свою очередь, если есть компьютер с процессором Core i7 3 ГГц и время работы решения жюри равно 0,2 секунды, то следует задать ограничение, равное 1 секунде.

4. Результатом решения всех предложенных задач является исходный текст программы на одном из языков программирования, принадлежащем основной или дополнительной группе (см. «Требования к организации и проведению регионального этапа Всероссийской олимпиады школьников по информатике в 2011/2012 учебном году»). Центральная предметно-методическая комиссия по информатике предоставляет региональному жюри материалы для проверки решений только для языков и сред программирования основной группы. При использовании организаторами регионального этапа языков и сред программирования дополнительной группы не гарантируется возможность получения полного решения олимпиадных задач регионального этапа. Более того, ответственность за проверку решений для языков и сред программирования дополнительной группы ответственность целиком и полностью несет региональное жюри.

5. Проверка решений участников осуществляется путем исполнения программы с входными данными, соответствующими каждому тесту из предложенного Центральной предметно-методической комиссией по информатике комплекта тестов с последующим анализом получаемых в результате этого выходных файлов.

6. Поскольку участники олимпиады должны сдавать на проверку решения в виде исходного текста программы на одном из допустимых языков программирования, то проверка решений каждого участника должна осуществляться в следующей последовательности:

- компиляция исходного текста программы;
- последовательное исполнение полученного exe-файла для файлов с входными данными, соответствующих тестам из набора тестов для данной задачи,

- сравнение результатов исполнения программы на каждом тесте с правильным ответом.

При компиляции исходного текста программы, которую участник сдал на проверку, необходимо учитывать следующее.

1) Жюри должно использовать вполне определенные командные строки для компиляции решений, о чем участников следует известить до начала тура. Эту информацию необходимо разместить в Памятке участнику, например, в следующем виде:

| **Компилятор** | **Командная строка** |
|---|---|
| Borland/Embarcadero Delphi 7.0 | dcc32 -cc <исходный файл> |
| Free Pascal 2.4.0 | fpc <исходный файл> |
| GNU C 4.4.2 (MinGW) | gcc -02 -x c –W1, --stack=67108864 <исходный файл> |
| Visual C++ 2005 | cl /02 /EHs /TP <исходный файл> |
| GNU C++ 4.4.2 (MinGW) | g++ -02 -x c++ –W1, --stack=67108864 <исходный файл> |

Жюри имеет право изменять команды компиляции решений в процессе проведения соревнований, но участники олимпиады должны быть обязательно проинформированы об этом перед началом тура.

2) Размер файла с исходным текстом программы не должен превышать **256** килобайт. Время компиляции программы не должно превышать **одной** минуты. В случае нарушения этих ограничений решение участника считается неправильным и никакие баллы за эту задачу участнику не начисляются. Информация об этих ограничениях также должна быть размещена в Памятке участнику.

3) При исполнении программы на каждом тесте, в первую очередь, жюри должно определить, нарушаются ли заданные в условии этой задачи ограничения на время работы программы на отдельном тесте и размер доступной программе памяти в процессе ее исполнения. В случае нарушения названных ограничений баллы за этот тест участнику не начисляются.

Если указанные выше ограничения в процессе исполнения программы с входными данными, соответствующими конкретному тесту, не нарушаются, то после завершения исполнения программы осуществляется проверка правильности полученного ответа. Эта проверка может осуществляться как путем сравнения полученного выходного файла с файлом выходных данных из тестового набора данных, так и с использованием предоставляемых центральной предметно-методической комиссией по информатике проверяющих программ, если для проверки решений участников используется

специализированная программная среда соревнований с возможность проверки решений в автоматическом режиме.

4) Все представленные на проверку решения участников сначала должны проходить предварительное тестирование на тесте или тестах из условия задачи. Если на этих тестах решение участника выдает правильный ответ, то тогда это решение принимается жюри на окончательную проверку на всех тестах из заданного набора тестов для этой задачи. В противном случае, решение участника считается неверным, и за него участнику не начисляются какие-либо баллы.

При проверке решений участников с использованием программной проверяющей системы процесс предварительной проверки осуществляется в течение тура по мере посылки решений на сервер соревнований. Результаты такой проверки сообщаются участнику сразу после ее завершения. В зависимости от возможностей проверяющей системы на окончательную проверку может приниматься либо последнее прошедшее предварительное тестирование решение одной и той же задачи, либо то, которое он должен указать. В любом случае, участник олимпиады должен быть проинформирован до начала тура, каким образом будет определяться решение, принимаемое системой для окончательной проверки. Эту информацию также следует разместить в Памятке участнику.

5) Окончательная проверка и оценивание решений участников может осуществляться либо после окончания тура, либо во время тура, если используемая жюри среда проведения соревнований позволяет это делать. Итоги такой проверки являются предварительными и доводятся индивидуально до сведения каждого участника только после завершения тура.

В заключение следует отметить, что результатом многократного исполнения программы участника с одними и теми же входными файлами должны быть одинаковые выходные файлы, вне зависимости от времени запуска программы и ее программного окружения. Региональное жюри вправе произвести неограниченное количество запусков программы участника и выбрать наихудший результат по каждому из тестов.

## 2.2. Характеристика тестов для каждой задачи

Комплект тестов для каждой задачи разрабатывался таким образом, чтобы региональное жюри могло в максимальной степени оценить все возможные типы алгоритмов, которые могут быть использованы в решениях участников, и продифференцировать полученные участниками решения по степени их корректности и

эффективности. В общем случае в комплекте тестов выделяются следующие группы тестов:

1) простые тесты;
2) тесты на частные случаи, позволяющие выявить особенности используемых алгоритмов;
3) общие тесты (достаточно случайные тесты, разные по размеру: от простых тестов до сложных);
4) тесты, проверяющие наличие эвристик в алгоритмах;
5) тесты максимальной размерности (тесты с использованием максимальных значений входных переменных, позволяющие оценить эффективность предложенных алгоритмов или их работоспособность при максимальной размерности задачи).

Количество тестов для каждой задачи различно и находится в диапазоне от 20 до 50. Файл с тестом называется «XY» и находится в каталоге tests. Тесты для каждой задачи пронумерованы от 1 до $N$, где $N$ — количество тестов к задаче. Файл с правильным ответом на соответствующий тест называется «XY.a». Здесь XY — двузначный номер теста, дополненный при необходимости ведущим нулем. Например, файл с седьмым тестом называется «07», а файл с пятнадцатым тестом — «15», файл с правильным ответом на седьмой тест называется «07.a», файл с правильным ответом на пятнадцатый тест — «15.a».

Количество тестов, подготовленных для каждой задачи, представлено в таблице ниже.

| **Задача** | **Количество тестов** |
|---|---|
| 1. Цапли | 20 |
| 2. Круглый стол | 20 |
| 3. Поврежденный XML | 20 |
| 4. Игра с числами | 25 |
| 5. Кондиционер | 20 |
| 6. Праздничный ужин | 50 |
| 7. Космический кегельбан | 50 |
| 8. Abracadabra | 20 |

Как уже было сказано выше, наборы тестов для каждой задачи разработаны таким образом, чтобы жюри могло в максимальной степени оценить все возможные типы алгоритмов, которые могут быть использованы в решениях участников, и продифференцировать полученные участниками решения по степени их корректности и

эффективности. Следует заметить, что правильное, но не эффективное решение задачи может набирать ориентировочно 30-70% баллов.

Ниже для каждой задачи описаны некоторые особенности оценивания возможных решений участников с использованием предложенных Центральной методической комиссией наборов тестов, которые будут полезны членам жюри и помогут лучше понять систему оценивания этих задач.

**Задача 1 «Цапли»**

Для проверки решений данной задачи выделены две группы тестов, отличающиеся ограничениями на числа $a$ и $b$:

1) тесты, для которых $1 \le a, b \le 1000$ (тесты 1 – 10);
2) тесты, для которых $1 \le a, b \le 10^9$ (тесты 11 – 20).

Вышеописанный набор тестов позволяет помимо полных решений оценивать частичные решения, выделенные в условии задачи,

**Задача 2 «Круглый стол»**

Набор тестов для данной задачи позволяет помимо полных решений оценивать следующие частичные решения задачи:

1) решения, верно работающие для случая: $2m = n$;
2) решения, верно работающие для случая: $2m < n$;
3) решения, верно работающие для случая: $2m > n$ и $n$ – четное число;
4) решения, верно работающие для случая: $2m > n$ и $n$ – нечетное число.

Каждое из этих частичных решений набирает по 25 баллов.

**Задача 3 «Поврежденный XML»**

Для проверки решений данной задачи выделены три группы тестов, отличающиеся видом полученной Петровым строки:

1) тесты, в которых исходная строка может быть преобразована в корректную XML-строку заменой одной буквы другой (тесты 1 – 10);
2) тесты, в которых исходная строка может быть преобразована в корректную XML-строку заменой других символов (тесты 11 – 20).

Вышеописанный набор тестов позволяет помимо полных решений оценивать частичные решения данной задачи, которые восстанавливают исходные строки, состоящие из тегов.

**Задача 4 «Игра с числами»**

Для проверки решений данной задачи выделены три группы тестов в зависимости от ограничений на количество фишек $n$:

1) тесты, для которых $1 < n \leq 15$ (тесты 1 – 10);
2) тесты, для которых $16 \leq n \leq 200$, (тесты 11-25).

Вышеописанный набор тестов позволяет помимо полных решений оценивать частичные решения данной задачи, в которых сохраняется подмножество чисел, соответствующих уже выбранным фишкам.

**Задача 5 «Кондиционер»**

Для проверки решений данной задачи используется одна группа тестов.

**Задача 6 «Праздничный ужин»**

Для проверки решений данной задачи используются две группы тестов.

Вышеописанный набор тестов позволяет помимо полных решений оценивать следующие частичные решения данной задачи:

1) решения, которые каждый раз создают новое блюдо;
2) решения, которые выдают правильный ответ только при $n_m = 1$.

**Задача 7 «Космический кегельбан»**

Для проверки решений данной задачи выделены четыре группы тестов. Группы отличаются ограничениями, накладываемыми на число $n$, и выполнением или невыполнением условия $v_x = 0$:

1) тесты, для которых $1 \leq n \leq 1000$ и $v_x = 0$ (тесты 1 – 10);
2) тесты, для которых $1 \leq n \leq 1000$ и $v_x \neq 0$ (тесты 11 – 20);
3) тесты, для которых $1000 < n \leq 200\,000$ и $v_x = 0$ (тесты 21 – 30).
4) тесты, для которых $1000 < n \leq 200\,000$ и $v_x \neq 0$ (тесты 31 – 50).

Вышеописанный набор тестов позволяет помимо полных решений оценивать следующие частичные решения данной задачи:

- решения, имеющие асимптотическую сложность алгоритма порядка $n^2$;
- решения, имеющие асимптотическую сложность алгоритма порядка $n$;
- решения, имеющие асимптотическую сложность алгоритма порядка $n \log n$.

**Задача 8. «Abracadabra»**

Для проверки решений данной задачи выделены две группы тестов в зависимости от ограничений на числа $n$ и $m$:

1) тесты, в которых $1 \leq n \leq 100$, $1 \leq m \leq 100$ (тесты 1 – 6);

2) тесты, в которых $100 < n$ или $100 < m$ (тесты 7 – 20);

Вышеописанный набор тестов позволяет помимо полных решений оценивать частичные решения данной задачи, в основу которых положен полный перебор всех пар «строка–запрос».

## 3. Система оценивания решений участников

Система оценивания решений каждой задачи основана на следующих положениях:

1. Решение каждой задачи оценивается из 100 баллов, то есть, максимальное количество баллов, которое участник может получить за полное решение каждой задачи, составляет 100 баллов.

2. Общая оценка за решение отдельной задачи конкретным участником складывается из суммы баллов, начисленных ему по результатам исполнения всех тестов из набора тестов для этой задачи в процессе окончательной проверки всех решений после тура.

3. Итоговый результат каждого участника подсчитывается как сумма полученных этим участником баллов за решение каждой задачи на первом и втором турах. С учетом того факта, что на каждом из двух туров предлагается по четыре задачи, то максимально возможное количество баллов, которое может набрать участник по итогам регионального этапа, составляет 800 баллов.

Для предложенных Центральной предметно-методической комиссией по информатике задач регионального этапа оценка для каждого теста из комплекта тестов для каждой задачи является одинаковой. В таблице ниже эти оценки представлены.

| Задача | Количество тестов | Оценка теста |
|---|---|---|
| 1. Цапли | 20 | 5 баллов |
| 2. Круглый стол | 20 | 5 баллов |
| 3. Поврежденный XML | 20 | 5 баллов |
| 4. Игра с числами | 25 | 4 балла |
| 5. Кондиционер | 20 | 5 баллов |
| 6. Праздничный ужин | 50 | 2 балла |
| 7. Космический кегельбан | 50 | 2 балла |
| 8. Abracadabra | 20 | 5 баллов |

Окончательные результаты проверки решений всех участников фиксируются в трех итоговых таблицах – для обучающихся 9-х, 10-х и 11-х классов. Каждая таблица представляет собой ранжированный список участников соответствующих классов, расположенных по мере убывания набранных ими баллов. Участники с одинаковыми

баллами располагаются в алфавитном порядке. На основании этих таблиц региональное жюри принимает решение о победителях и призерах регионального этапа олимпиады по каждому классу с учетом квоты, установленной организатором регионального этапа.

## 4. Автоматизация процесса проверки и оценивания решений участников

Существуют различные способы проверки решений участников. Если по условию задачи ее решением должна быть программа, то самый простой способ, но в то же время самый трудоемкий, заключается в последовательном запуске проверяемой программы на каждом тесте из заданного комплекта тестов для этой задачи. Для этого способа вполне достаточно иметь для каждого теста файл с входными данными и файл с соответствующими выходными данными. Если учесть, что для каждой задачи эти файлы предоставляются региональному жюри центральной предметно-методической комиссией по информатике, то члены жюри вполне могут справиться с задачей проверки решений участников таким «ручным» способом при наличии достаточного количества членов жюри.

Конечно, описанный способ достаточно трудоемкий, но тот факт, что решения участников сначала проверяются на одном или двух тестах из условия задачи, и только в случае успешного прохождения этих тестов решение далее проверяется на всех тестах из заданного набора, в определенной степени уменьшает объем необходимой работы. Тем не менее, выходом из создавшегося положения является автоматизация процесса проверки решений участников.

В настоящее время во многих регионах уже успешно используются специализированные системы проведения различных соревнований по информатике. Они используются как при предварительной проверке решений участников во время тура, так и при окончательной проверке, которая осуществляется после завершения тура. Более того, уже де-факто сформировались вполне определенные к ним требования.

В частности, в процессе предварительной проверки решений участников, представленных в виде программ, такие системы должны последовательно выполнять следующие действия:

1) Скомпилировать программу участника, используя приведенную в Памятке участнику команду для соответствующего языка программирования. Если компиляция программы участника завершается неудачно, участнику сообщается результат «Ошибка компиляции». Возможно предоставление участнику вывода компилятора в стандартный поток вывода и стандартный поток ошибок. Если компиляция завершилась успешно, то далее программа проверяется на тестах из примера.

2) Осуществлять последовательную проверку программы участника на всех тестах из примера. Проверка на одном тесте осуществляется следующим образом. В пустой каталог копируется исполняемый файл программы участника и тестовый входной файл. Тестовый файл должен иметь имя, указанное в условии задачи. Далее программа участника запускается, и проверяющая система отслеживает соблюдение программой существующих ограничений, связанных с запретом на создание каталогов и временных файлов при работе программы, осуществление чтения и записи векторов прерываний, а также любое использование сетевых средств и выполнение других действий, нарушающих работу самой проверяющей системы.

3) Обеспечивать контроль времени работы программы участника и объема используемой памяти. Если время работы программы превысило ограничение, указанное в условии задачи, выполнение программы участника прерывается и участнику отправляется сообщение «Превышено время работы». Если количество используемой памяти превысило ограничение, указанное в условии задачи, то выполнение программы участника также прерывается и участнику отправляется сообщение «Превышен максимальный объем используемой памяти».

4) Проверить, создала ли программа участника и самостоятельно обработала исключительную ситуацию (exception). Если программа участника создала и самостоятельно не обработала исключительную ситуацию (exception), выполнение программы участника прерывается и участнику отправляется сообщение «Ошибка времени исполнения».

5) Проверить, завершила ли программа участника работу с нулевым кодом возврата. Если программа участника завершила работу с ненулевым кодом возврата, участнику отправляется сообщение «Ошибка времени исполнения».

6) Проверить, создала ли программа участника в каталоге, в котором она была запущена, выходной файл с именем, указанным в условии задачи, если программа участника завершила работу за отведенный период времени, не превысила максимальный объем памяти и завершила работу с нулевым кодом возврата. Если файл с указанным именем не найден, участнику отправляется сообщение «Ошибка формата выходных данных». Если выходной файл создан, то осуществляется проверка его корректности. Для этого используется соответствующая проверяющая программа, которая предоставляется организаторам регионального этапа центральной предметно-методической комиссией по информатике.

7) Сообщить участнику о результатах проверки его программы. Если программа участника выдает правильный ответ на всех тестах из примера, то она может быть

принята на окончательную проверку. В этом случае участнику отправляется сообщение «Принято на проверку», а тестирующая система запоминает решение участника как последнее принятое решение по данной задаче. В противном случае участнику отправляется сообщение в соответствии с описанными выше правилами. При этом участнику помимо типа ошибки сообщается номер теста из примера, на котором произошла ошибка.

При окончательной проверке решений участников, представленных в виде программ, которая осуществляется после тура, программная система проведения олимпиад по информатике должна проверить на основных тестах последнее принятое на проверку решение участника по каждой задаче. Выполняемые системой функции в этом случае во многом повторяют вышеописанные. Однако при возникновении любой ошибки вместо отправки участнику сообщения, система просто помечает тест как не пройденный. Кроме того, по результатам окончательной проверки система начисляет участнику баллы за успешно пройденные тесты.

Для упрощения процесса использования систем, автоматизирующих проверку решений участников, Центральная предметно-методическая комиссия предоставляет в распоряжении регионального жюри специально разработанные для каждой задачи проверяющие программы.

Проверяющая программа для каждой задачи находится в папке с номером этой задачи либо в файле «check.dpr», либо в файле «check.cpp». Для компиляции файла «check.dpr» следует использовать компилятор Borland Delphi. Кроме того, при ее исполнении используется библиотека для проверяющих программ testlib, которая содержится в папке «lib» в файле «testlib.pas». Для компиляции файла «check.cpp» можно использовать GNU C++ или Visual C++. Используемая в этом случае библиотека «testlib.h» также содержится в папке «lib».

Запуск проверяющей программы осуществляется следующим образом. В пустой каталог необходимо скопировать проверяющую программу, входной файл, ответ для которого следует проверить (input), выходной файл, созданный программой участника (output) и файл с правильным ответом (answer). После этого проверяющая программа запускается с тремя параметрами командной строки — входной файл, выходной файл и файл с правильным ответом.

Программа завершает свою работу с одним из трех возможных кодов возврата: 0 — ответ участника является правильным; 1 — ответ участника удовлетворяет формату вывода, но является неправильным, в этом случае участнику отправляется сообщение «Неверный ответ»; 2 — ответ участника не удовлетворяет формату вывода, в этом случае

участнику отправляется сообщение «Ошибка формата выходных данных». Файл «result» после завершения программы будет содержать сформированный проверяющей программой комментарий о причинах, по которым был выдан соответствующий отклик. Этот комментарий не должен сообщаться участникам и служит только для справки членам регионального жюри.

Проверяющая программа check.exe используется как на этапе предварительной проверки, так и на этапе окончательной проверки и оценки решений участников. Отличие заключается в том, что при окончательной проверке при возникновении любой ошибки вместо формирования участнику сообщения, система просто помечает тест как не пройденный, и по результатам проверки участнику начисляются баллы за успешно пройденные тесты в соответствии с системой оценивания, разработанной центральной предметно-методической комиссии по информатике.

В заключении хотелось бы отметить, что разработать простейшую программную систему, позволяющую осуществлять окончательную проверку решений участников в автоматическом режиме, является не такой уж сложной задачей и по силам любой региональной предметно-методической комиссии по информатике. Наличие такой системы позволит на должном уровне осуществлять проверку решений участников не только на региональном, но и на муниципальном и школьном этапах Олимпиады по информатике в субъекте РФ.

## 5. Краткие методические рекомендации по решению задач

При разработке задач для регионального этапа олимпиады по информатике Центральная предметно-методическая комиссия исходила из того, что все задачи должны быть оригинальными, быть разнообразными по тематике и не требовать для своего решения специальных знаний.

При определении уровня сложности задач разработчики исходили из того, что комплект задач должен содержать как задачи, доступные многим участникам регионального этапа, так и задачи, позволяющие проявить себя наиболее сильным участникам. В этой связи количество задач на каждом туре равно четырем, и как минимум две задачи из них такой сложности, что большинство школьников могут их решить, включая и школьников младших классов. Более того, все задачи являются многоуровневыми и предполагают наличие как полных, так и частичных решений, что также будет способствовать тому, что ни одна задача из предложенного комплекта не останется без внимания участников, а сильным участникам позволит продемонстрировать все свои лучшие качества.

Сказанное подтверждает общая характеристика всех задач, представленная в таблице ниже.

| Задача | Тематика | Алгоритмическая сложность | Техническая сложность |
|---|---|---|---|
| 1. Цапли | Математические основы информатики | низкая | низкая |
| 2. Круглый стол | Математические основы информатики | средняя | низкая |
| 3. Поврежденный XML | Техника программирования | низкая | высокая |
| 4. Игра с числами | Динамическое программирование, игры на графах | высокая | высокая |
| 5. Кондиционер | Техника программирования | низкая | низкая |
| 6. Праздничный ужин | Математические основы информатики | средняя | средняя |
| 7. Космический кегельбан | Геометрия | высокая | средняя |
| 8. Abracadabra | Алгоритмы на строках | высокая | высокая |

Далее приведены краткие методические указания по решению каждой задачи, которые помогут членам жюри выявить те или иные особенности существующих решений, осмысленно подойти к проверке решений участников и подготовиться к разбору задач, который необходимо провести после завершения второго тура и получения результатов предварительной проверки решений всех участников.

**Задача 1. «Цапли»**

Данная задача является наиболее простой в комплекте задач для первого тура, и ее решение основано на следующем подходе. Обозначим через k возможное количество цапель. Понятно, что если Петя увидел $a$ ног, то цапель будет не больше, чем $a$, поскольку каждая цапля стоит хотя бы на одной ноге. С другой стороны, их должно быть не меньше, чем $a/2$, иначе ног у них было бы меньше, чем $a$. Таким образом, получается соотношение:

$$a/2 \le k \le a.$$

Аналогичными рассуждениями можно придти к соотношению:

$$b/2 \le k \le b.$$

Поскольку по условию задачи хотя бы одно решение данной системы неравенств существует, получаем:

$$\max\{a,\ b\}/2 \le k \le \min\{a, b\}.$$

Если $\max\{a,\ b\}/2$ – нецелое число, его нужно округлить до большего целого числа. Это можно сделать, например, с помощью выражения $(\max\{a, b\} + 1)\ //\ 2$, где // обозначает целочисленное деление. Соответствующий фрагмент программы на языке C++ приведен ниже:

```
cin >> a >> b;
cout << (max(a, b) + 1) / 2 << " " << min(a, b);
```

**Задача 2. «Круглый стол»**

Решение данной задачи основано на анализе возможных вариантов размещения участников за круглом столом и выявлении закономерностей, характерных для оптимальных размещений. Прежде всего попробуем рассадить вокруг стола какое-нибудь количество мальчиков и девочек, так чтобы все они принимали активное участие в обсуждении. Начнем с того, что посадим за стол одного мальчика. Если слева и справа от него посадить девочек (GBG), то все они будут активными участниками круглого стола. При добавлении еще двух участников клуба с сохранением максимальной активности рядом с каждой девочкой нужно посадить еще по девочке (GGBGG). Если рядом с новыми девочками посадить еще по одному мальчику (BGGBGGB), то опять получим ситуацию, когда все участники будут активными.

Можно заметить, что при рассаживании детей по схеме:

B – GG – B – GG – B – GG – B – … – GG,

получается, что в итоге *все* мальчики и девочки будут принимать активное участие в обсуждении. При этом количество девочек окажется вдвое больше количества мальчиков.

Покажем теперь, как будут выглядеть другие возможные оптимальные размещения участников круглого стола.

1. Количество девочек больше, чем удвоенное количество мальчиков ("лишние" девочки подчеркнуты):

BGGBGGBGG … BGGBGGGGG … GG

2. Количество девочек четно и меньше, чем удвоенное количество мальчиков:

BGGBGGBGG … BGGBGGBBB … BB

3. Количество девочек нечетно и меньше, чем удвоенное количество мальчиков:

BGGBGGBGG … BGGBGGBBB … BG

Докажем, что в этих примерах количество участвующих в обсуждении детей максимально возможное.

1. Пусть $n > 2m$. Поскольку каждый мальчик может способствовать активному участию в обсуждении не более двух девочек, то всего удастся привлечь к активному обсуждению не более $2n$ девочек. Таким образом, в этом случае в обсуждении будут участвовать все $n$ мальчиков и ровно $2n$ девочек.

2. Пусть $n = 2k < 2m$. В этом случае в обсуждении будут участвовать *все* $2k$ девочек и $(k – 1)$ мальчиков. Пусть мы можем пересадить школьников так, чтобы $(k + x)$ мальчиков были активны. Рассмотрим пары рядом сидящих детей разного пола. Каждая активная девочка входит ровно в одну такую пару, каждая неактивная девочка может входить в две такие пары. Пусть $(k + x)$ мальчиков активны $(x > 0)$, тогда пар мальчик-девочка (или девочка-мальчик) должно быть не менее $2(k + x) = 2k + 2x$, откуда неактивных девочек должно быть хотя бы $2x$. В этом случае общее количество активных детей равно $(k + x) + (2k – 2x) = 3k – x$. Это число больше $(3k – 1)$ лишь при $x = 0$. Таким образом, если и можно рассадить детей лучше, то в этом случае активными будут ровно $k$ мальчиков и все $2k$ девочек. Докажем, что и этого сделать нельзя.

Действительно, чтобы все девочки были активны, они должны сидеть парами (рядом с каждой девочкой должна быть девочка), с другой стороны, чтобы $k$ мальчиков были активны, они должны сидеть между двумя девочками каждый, то есть, рядом с девочками должны сидеть только активные мальчики. Таким образом, получается, что каждый активный участник должен сидеть рядом с двумя другими активными участниками, то есть, никакой активный участник не должен сидеть рядом с неактивным, что невозможно, если все школьники сидят за круглым столом.

3. Пусть $n = 2k + 1 < 2m$. В этом случае активны $k$ мальчиков и $2k$ девочек: всего $3k$ школьников. Пусть при некоторой рассадке $(k + x)$ мальчиков окажутся активными $(x > 0)$. Тогда пар мальчик-девочка (или девочка-мальчик) должно быть не менее $2(k + x) = 2k + 2x$, откуда следует, что неактивных девочек должно быть хотя бы $2x$. Поскольку общее количество девочек нечетно, а количество активных девочек должно быть четно, то количество неактивных девочек будет нечетным, а потому не меньше $(2x + 1)$. Следовательно, общее количество активных детей равно $(k + x) + (2k + 1 – (2x + 1)) = 3k – x$. Это число не больше $3k$, следовательно, приведенный пример является оптимальным.

Используя результаты проведенного анализа, уже не трудно далее получить решение исходной задачи.

**Задача 3. «Поврежденный XML»**

Первая идея, которая приходит после внимательного прочтения условия данной задачи, связана с использованием полного перебора для ее решения. Осталось только проанализировать возможность его использования для заданной размерности входной строки.

Несложные расчеты показывают, что поскольку ограничения на размер входной строки в этой задаче небольшие, ее действительно можно решать полным перебором. В частности, будем пытаться исправлять каждый символ по порядку на все допустимые символы (латинские буквы, символы '<', '>', '/'), до тех пор, пока не получится корректная XML-строка.

Реализовать такой перебор является делом техники. Осталась только разобраться с тем, каким образом осуществляется проверка, является ли строка корректной XML-строкой.

Для решения этой задачи выполним следующие шаги.

1) Убедимся, что строка начинается с символа '<', а заканчивается символом '>'. Если это так, то удалим начальный и конечный символы.

2) Выделим в строке пары символов '> <', удаляя при этом эти символы. Получим список строк-тегов (открывающих или закрывающих).

3) Проверим, что в этих тегах нет символов '<' или '>', а также нет символов '/' не на первой позиции.

4) Проверим, что эти теги образуют "правильную скобочную последовательность".

Опишем более подробно выполнение последнего шага. Сформируем стек тегов (строк), изначально пустой. Далее будем перемещаться по тегам слева направо, при этом рассмотрим два случая:

1. Очередной тег – открывающий: в этом случае добавим новый тег в конец стека.
2. Очередной тег – закрывающий: в этом случае необходимо рассмотреть две ситуации:

   а) стек пустой (значит, найден закрывающий тег, которому не соответствует открывающий тег; следовательно, строка не является корректной XML-строкой);

   б) стек не пустой (тогда необходимо извлечь из стека последний элемент; если этот тег не соответствует нашему закрывающему тегу, то XML-строка некорректная).

Если пройден весь список тегов, и ни на каком шаге не было установлено, что строка некорректна, то осталось лишь проверить, что стек в конце оказался пуст, то есть, "лишние" открывающиеся теги отсутствуют.

С учетом приведенного анализа уже не сложно реализовать решение исходной задачи.

**Задача 4. «Игра с числами»**

Начнем решение данной задачи с того, что обозначим через *first* число, выбранное на первом ходу. Тогда каждое очередное число нужно выбирать так, чтобы НОД разностей всех выбранных чисел с первым числом был строго больше 1. Заметим, что если $d$ – это НОД разностей выбранных чисел с первым числом, то оставшиеся числа делятся на две группы:

- числа, при выборе которых на последующих ходах НОД не изменится;
- числа, которые при их выборе уменьшат НОД.

С одной стороны, для дальнейшей игры нам не важно, какие именно числа находятся в первой группе (важно лишь их количество). С другой стороны, чтобы определить, принадлежит ли некоторое число $a$ второй группе, необходимо проверить, делится ли $|a - first|$ на $d$.

С учетом сказанного, можно сделать вывод, что позицию в игре, начавшейся с выбора числа *first*, следует задавать двумя переменными: количеством сделанных ходов *num* и НОД $d$ разностей всех выбранных чисел с первым числом. Кроме того, для каждого возможного первого хода будем решать задачу отдельно.

Поскольку количество теоретически возможных значений НОД может быть довольно велико, то задачу лучше всего решать с использованием рекурсии с запоминанием ("ленивым" динамическим программированием).

Анализ позиций ($num$, $d$) в этом случае будет заключаться в следующем.

1) Если $num = n$, то позиция проигрышная (все числа уже выбраны; ход сделать нельзя).
2) Подсчитаем, количество чисел *count* в изначальном наборе чисел, разности которых с числом *first* делятся на $d$:
   a) рассмотрим сначала числа, выбор которых уменьшает $d$, то есть такие числа $x$, для которых $|x - first|$ не делится на $d$, и при этом НОД($d$, $|x - first|$) должен быть больше 1, в противном случае выбрать число $x$ уже нельзя. Позиция

($num, d$) будет выигрышной, если ($num + 1$, НОД ($d, |x – first|$) – проигрышная.

б) если $count > num$, то можно выбрать число, делящееся на $d$, получив при этом позицию ($num + 1, d$). Если она проигрышная, то ($num, d$) – выигрышная.

3) В противном случае позиция ($num, d$) – проигрышная.

Осталось отметить еще один момент: если $num = 1$, то выбрано пока только число *first*. В этом случае все рассуждения останутся верными, если положить $d = 0$.

**Задача 5. «Кондиционер»**

Приведём два возможных подхода к решению данной задачи.

Первый подход. Заметим, что искомое решение зависит только от режима работы кондиционера и взаимного расположения чисел $t_{room}$ и $t_{cond}$. В частности, все возможные комбинации можно представить в виде следующей таблицы:

| | **freeze** | **heat** | **auto** | **fan** |
|---|---|---|---|---|
| $t_{room} < t_{cond}$ | $t_{room}$ | $t_{cond}$ | $t_{cond}$ | $t_{room}$ |
| $t_{room} = t_{cond}$ | $t_{room}$ | $t_{cond}$ | $t_{cond}$ | $t_{room}$ |
| $t_{room} > t_{cond}$ | $t_{cond}$ | $t_{room}$ | $t_{cond}$ | $t_{room}$ |

С учетом сказанного фрагмент программы на языке C++, реализующий идею этого решения, будет выглядеть следующим образом:

```
cin >> troom >> tcond;
cin >> s;
if (s=="freeze") || (s=="heat" && troom>tcond) ||
(s=="freeze") {
    cout << troom;
} else {
    cout <<tcond;
}
```

Второй подход. Заметим, что при каждом режиме работы кондиционер реализует некоторую функцию, которая вычисляет результат по двум аргументам $t_{room}$ и $t_{cond}$. В режиме "freeze" кондиционер реализует функцию $\min\{x, y\}$, в режим "heat" – функцию $\max\{x, y\}$, в режиме "auto" – функцию $f(x, y) = y$ (возвращает второй аргумент), а в режиме "fan" – функцию $g(x, y) = x$ (возвращает первый аргумент).

Фрагмент программы на языке C++, реализующий идею этого решения, приведен ниже:

```
if (s == "freeze") cout << ((troom > tcond) ? tcond : troom);
if (s == "heat") cout << ((troom < tcond) ? tcond : troom);
```

```
if (s == "fan") cout << troom;
if (s == "auto") cout << tcond;
```

**Задача 6. «Праздничный ужин»**

Пусть некоторому сотруднику компании предложили $p$ вариантов ужина, а следующему – $q$. В этом случае справедливы следующие соотношения:

$$p = a_1 \times a_2 \times \ldots \times a_i \times \ldots \times a_k, \quad q = a_1 \times a_2 \times \ldots \times (a_i - 1) \times \ldots \times a_k, \quad p / q = a_i / (a_i - 1).$$

Из этого следует, что $a_i = p / (p - q)$, то есть, когда сотруднику компании предложили $p$ вариантов ужина, одно из блюд предлагалось в $p / (p - q)$ вариантах.

Из этих рассуждений видно, что постепенно анализируя входные данные, можно получать новую информацию о количестве блюд того или иного типа. Для этого определим два массива: в одном массиве (*start*) будем хранить изначальное количество каждого из типов блюд (сначала массив будет пустой, а по мере получения новой информации он будет расширяться). Во втором массиве (*current*) в элементе с тем же номером будет храниться текущее количество вариантов того же типа блюда (на самом деле решений может быть несколько: мы будем хранить количества, соответствующие одному из них).

Пусть в некоторый момент обнаруживается, что было $a_i$ вариантов некоторого типа блюда. Тогда возможны две ситуации:

1) в массиве *current* уже есть число $a_i$. Тогда считается, что это то же самое блюдо, и этот элемент массива *current* уменьшается на 1.
2) в массиве *current* нет числа $a_i$. Это означает, что получена информация о блюде, про которое ничего ранее не было известно. Тогда в массив *start* добавляется число $a_i$, а в массив *current* – число $(a_i - 1)$.

Заметим, что если бы в первом пункте создалось новое блюдо, то в итоге можно было бы получить количество блюд, большее, чем указано в условии.

После обработки всех входных данных получается массив *start*, длина которого меньше или равна количеству блюд $k$. Если она равна $k$, то задача решена. Если длина массива *start* меньше $k$, то необходимо рассмотреть количество вариантов $r$, которое было предложено на выбор последнему сотруднику компании. Поскольку в массиве *current* к этому моменту уже содержится информация о количестве некоторых типов блюд, то разделим $r$ на произведение элементов массива *current* и будем считать, что появился еще один тип блюд с таким изначальным количеством. Если же и теперь количество типов блюд меньше, чем требуется, то добавляются оставшиеся типы блюд с изначальным количеством, равным 1.

В качестве иллюстрации сказанного рассмотрим следующий пример. Пусть последнему сотруднику компании остался выбор из 4-х заказов, всего блюд было 5, и в массиве *start* содержатся числа 2 и 3. Тогда можно считать, что изначально был выбор из 2-х блюд первого типа, 3-х блюд второго типа, 4-х блюд третьего типа, одного блюда четвертого типа и одного блюда пятого типа.

**Задача 7. «Космический кегельбан»**

В результате формализации условия данной задачи получим следующую ее формулировку. Дано несколько кругов радиуса $r$ (кегли) и полоса шириной $2q$ (траектория движения шара). Требуется подсчитать, сколько окружностей имеют хотя бы одну общую точку с этой полосой.

Эту формулировку можно уточнить, заменив окружности точками, а полосу шириной $2q$ на полосу шириной $2(q + r)$. Тогда исходная задача сведется к эквивалентной: необходимо определить, сколько точек (центров кеглей) лежат в расширенной полосе, то есть, находятся на расстоянии не более $(q + r)$ от прямой, по которой движется центр шара. При этом, поскольку по условию задачи каждая точка шара лежит изначально ниже любой точки каждой кегли, то можно рассматривать не часть полосы выше шара, а всю полосу, и на ответ это не повлияет.

Запишем в виде неравенства условие принадлежности точки полосе. Пусть $A$ – начальное положение центра шара, вектор $AB$ имеет координаты $(v_x, v_y)$, $P$ – центр некоторой кегли. Тогда расстояние от точки $P$ до прямой, совпадающей с вектором $AB$, будет равно $|[AP, AB]| / |AB|$ (с использованием квадратных скобок обозначено псевдоскалярное произведение векторов), а условие принадлежности точки полосе запишется так:

$$|[AP, AB]| / |AB| \le q + r.$$

С учетом сказанного рассмотрим различные подходы к решению данной задачи, которые позволяют получить как некоторые частичные, так и полное решение. При этом будем использовать принцип «от простого к сложному».

Первое решение, которое имеет асимптотическую сложность алгоритма порядка $n^2$, заключается в следующем. Сначала проверим отдельно принадлежность нашей полосе центра $P(x_p, y_p)$ каждой кегли. Для этого запишем приведенное выше неравенство с использованием координат соответствующих точек. Пусть точки $A$ и $B$ имеют координаты $(x, y)$ и $(x + v_x, y + v_y)$ соответственно ($B$ – точка на прямой, по которой движется центр шара). Тогда неравенство примет вид:

$$|(x_P - x)\, v_y - (y_P - y)\, v_x)| / \sqrt{v_x^2 + v_y^2} \le q + r.$$

Далее помножив обе части неравенства на знаменатель и возведя их в квадрат, получим следующее неравенство:

$$((x_P - x)\, v_y - (y_P - y)\, v_x))^2 \le (v_x^2 + v_y^2)\, (q + r)^2.$$

Теперь, подставляя последовательно в это неравенство центры всех кеглей, можно легко найти искомый ответ.

Следует заметить, что полученное неравенство предполагает использование в процессе его вычисления только целых чисел. С одной стороны, это избавляет нас от всех возможных проблем, связанных с точностью вычислений, но с другой стороны, может создать проблемы с переполнением, так как правая часть неравенства с учетом имеющихся в задаче ограничений может достигать значения, равного $((10^{12})^2 + (10^{12})^2) * (10^9)^2 = 10^{32}$, что превосходит границу чисел, которые можно хранить даже в 64-битных целых типах.

Время работы данного алгоритма будет определяться общим количеством кеглей, которое равно $1 + 2 + \ldots + n = n(n + 1)/2 \sim n^2$. Поэтому асимптотическая сложность такого алгоритма порядка $n^2$, и получаемое в этом случае решение набирает 40 баллов.

Второе решение, которое имеет асимптотическую сложность алгоритма порядка $n$, основано на следующем выводе: для решения исходной задачи достаточно найти для каждого горизонтального ряда кеглей самую левую кеглю и самую правую кеглю, которые собьет шар. Для этого необходимо решить относительно $x_p$ для каждого значения $y_p$ неравенство:

$$((x_P - x)\, v_y - (y_P - y)\ \ v_x))^2 \le (v_x^2 + v_y^2)\, (q + r)^2$$

Это квадратичное неравенство и решением его является отрезок с концами:

$$left = k(y_p - y) - c + x,$$

$$right = k(y_p - y) + c + x,$$

где $k = v_x/v_y$, $c = (q + r)\, \sqrt{v_x^2 + v_y^2}\ / v_y$.

Теперь осталось найти самый левый и самый правый центр кегли на этом отрезке, то есть, самое левое/правое четное или нечетное целое число (четность зависит от четности $y_p$), а также ограничить координату по модулю числом $y_p$.

Количество сбитых кеглей в горизонтальном ряду легко вычислить по формуле:

$$\max\{(right - left)\ //\ 2 + 1, 0\}.$$

В процессе вычисления значений величин *left* и *right* может оказаться, что $right < left$. В этом случае шар не сбивает ни одной кегли в данном ряду.

Третье решение, которое имеет асимптотическую сложность алгоритма порядка $n \log n$, заключается в следующем. Перепишем неравенство, используемое в первом решении, таким образом, чтобы избавиться от модуля. В результате этого получим:

$$-(q+r)\sqrt{v_x^2+v_y^2} \le ((x_P-x)\,v_y-(y_P-y)\,v_x) \le (q+r)\sqrt{v_x^2+v_y^2}.$$

Не сложно заметить, что при движении по горизонтальному ряду кеглей слева направо средняя часть этого неравенства возрастает ($x_p$ возрастает, $v_y = \text{const} > 0$). Следовательно, искать целочисленные решения этого неравенства можно бинарным поиском. В частности, найдем самое левое и самое правое целое число, принадлежащее отрезку, и прибавим/вычтем 1, если число окажется ненадлежащей четности.

Но этого не достаточно, чтобы достигнуть цели, поскольку в неравенстве присутствует нецелое число в виде квадратного корня, а при возведении его в квадрат средняя часть неравенства перестает быть возрастающей функцией. Для решения этой проблемы воспользуемся следующим. Перепишем неравенство

$$((x_P-x)\,v_y-(y_P-y)\,v_x) \le (q+r)\sqrt{v_x^2+v_y^2},$$

не нарушая эквивалентность и пользуясь тем, что $q + r > 0$. В результате этого получим:

$$((x_P-x)\,v_y-(y_P-y)\,v_x) < 0 \quad \text{or} \quad ((x_P-x)\,v_y-(y_P-y)\,v_x))^2 \le (v_x^2+v_y^2)(q+r)^2.$$

Для проверки этого условия уже требуются только вычисления в целых числах, а значит, цель достигнута. Аналогично переписывается и левая часть двойного неравенства. Однако и в этом случае вычисления нельзя провести в 64-битном целом типе из-за больших ограничений в условии задачи. Тем не менее, преимущество этого решения заключается в том, что оно, оставаясь достаточно быстрым, оперирует лишь целыми числами, а значит, лишено возможных проблем, возникающих из-за ошибок, связанных с округлением вещественных чисел при вычислениях.

**Задача 8. «Abracadabra»**

Основная идея решения данной задачи заключается в следующем. Преобразуем данный нам словарь к такому виду, чтобы в нем можно было быстро искать слово с нужным супрефиксом. Для этого каждую строку $s[0..n-1]$ из словаря преобразуем в строку следующего вида:

$$s[0]\ s[n-1]\ s[1]\ s[n-2]\ s[2]\ s[n\text{-}3]\ \ldots\ s[n-1]\ s[0].$$

В результате таких преобразование, например, слово 'table' превратится в строку 'tealbblaet'.

Далее отсортируем обновленный таким образом словарь в лексикографическом порядке (по алфавиту). Заметим, что теперь все слова с одинаковым супрефиксом будут идти подряд.

С каждой из строк-образцов проделаем точно такое же преобразование. Теперь для каждой обновленной строки-образца *s* найдем бинарным поиском самую первую позицию (*left*) в словаре, куда ее можно поместить, не нарушая лексикографический порядок. Далее

возьмем строку, полученную из *s* заменой последнего символа на следующий в таблице ASCII символ, которая будет иметь вид:

s[n] = chr(ord(s[n] + 1))),

и найдем самую левую позицию в словаре, куда его можно поместить (*right*). Заметим, что все слова, имеющие супрефикс *s*, лежат в промежутке [*left*, *right*), поэтому искомое количество слов будет равно (*right* – *left*).

Сказанное можно продемонстрировать на следующем примере. Пусть словарь после описанного выше преобразования имеет вид:

*s*[0] = 'aa';

*s*[1] = 'aaaa';

*s*[2] = 'aaaabbaaaa';

*s*[2] = 'abba';

*s*[3] = 'acbbca';

а строка-образец после аналогичного преобразования имеет вид 'aaaa'. Тогда значение *left* будет равно самой левой позиции, куда можно поместить строку 'aaaa', то есть 1, а значение *right* будет равно самой левой позиции, куда можно поместить строку 'aaab', то есть 3. С учетом этого, искомый ответ в данной задаче будет равен (right – left) = (3 – 1) = 2.

## Список рекомендуемой литературы


1. Алексеев В.Е., Таланов В.А. Графы и алгоритмы. Структуры данных. Модели вычислений. – М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2006. – 320 с. – (Серия «Основы информационных технологий»)
2. Андреева Е.В., Босова Л.Л., Фалина И.Н. Математические основы информатики. Элективный курс: Учебное пособие. – М.: БИНОМ. Лаборатория Знаний, 2007. – 312 с.
3. Арсак Ж. Программирование игр и головоломок. – М.: Наука, 1990. – 224 с.
4. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. – М.: Издательский дом «Вильямс», 2000. – 384 с.
5. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. — Пер. с англ. — М.: Мир, 1979. — 536 с.
6. Бентли Д. Жемчужины творчества программистов: пер. с англ. – М.: Радио и связь, 1990. – 224 с.
7. Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ. – М.: Мир, 1985.
8. Вирт Н. Алгоритмы и структуры данных. Пер. с англ. М.: Мир, 1989. – 360 с.
9. Гасфилд Дэн. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. И.В.Романовского. – СПб.: Невский Диалект; БХВ Петербург, 2003. – 654 с.
10. Джонстон Г. Учитесь программировать. – М.: Финансы и статистика, 1989. – 336 с.
11. Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И. Лекции по теории графов. – М.: Наука, 1990. – 384 с.
12. Задачи по программированию /С.М. Окулов, Т.В. Ашихмина, Н.А. Бушмелева и др.; Под ред. С.М. Окулова. – М.: БИНОМ. Лаборатория знаний, 2006. – 820 с.
13. Златопольский Д. М. Программирование: типовые задачи, алгоритмы, методы. – М.: БИНОМ. Лаборатория знаний, 2007. – 223 с.
14. Кирюхин В.М. Информатика. Всероссийские олимпиады. Выпуск 1. – М.: Просвещение, 2008. – 220 с. – (Пять колец).
15. Кирюхин В.М. Информатика. Всероссийские олимпиады. Выпуск 2. – М.: Просвещение, 2009. – 222 с. – (Пять колец).
16. Кирюхин В.М. Информатика. Всероссийские олимпиады. Выпуск 3. – М.: Просвещение, 2011. – 222 с. – (Пять колец).

17. Кирюхин В.М. Информатика. Международные олимпиады. Выпуск 1. – М.: Просвещение, 2009. – 239 с. – (Пять колец).
18. Кирюхин В.М., Окулов С. М. Методика решения задач по информатике. Международные олимпиады. – М.: БИНОМ. Лаборатория знаний, 2007. – 600 с.
19. Кнут Д. Искусство программирования для ЭВМ. Т. 1-3. – М., СПб., Киев: Вильямс, 2000.
20. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 1999. – 960с.
21. Кристофидес Н. Теория графов. Алгоритмический подход. – М.: Мир, 1978. – 432 с.
22. Липский В. Комбинаторика для программистов. – М.: Мир, 1988. – 77 с.
23. Майерс Г. Искусство тестирования программ. Пер. с анг. под ред. Б.А. Позина. – М.: Финансы и статистика, 1982. – 176 с.
24. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики. – СПб.: БХВ-Петербург, 2003. – 560 с.
25. Окулов С.М. Программирование в алгоритмах. – М.: БИНОМ. Лаборатория знаний. 2002. – 341 с.
26. Окулов С.М. Дискретная математика. Теория и практика решения задач по информатике: учебное пособие. – М.: БИНОМ. Лаборатория знаний. 2008. – 422 с.
27. Окулов С.М. Алгоритмы обработки строк: учебное пособие. – М.: БИНОМ. Лаборатория знаний, 2009. – 255 с.
28. Окулов С.М. Абстрактные типы данных. – М.: БИНОМ. Лаборатория знаний, 2009. – 250 с.
29. Окулов С.М. Динамическое программирование. – М.: БИНОМ. Лаборатория знаний, 2011. – 260 с.
30. Просветов Г.И. Дискретная математика: задачи и решения: учебное пособие. – М.: БИНОМ. Лаборатория знаний. 2008. – 222 с.
31. Рейнгольд Э. Комбинаторные алгоритмы: теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део. – М.: Мир, 1980. – 476 с.
32. Столяр С.Е., Владыкин А.А.. Информатика. Представление данных и алгоритмы. – СПб.: Невский Диалект; М.: БИНОМ. Лаборатория знаний. 2007. –382 с.
33. Уэзерелл Ч. Этюды для программистов. – М.: Мир, 1982. – 288 с.
34. Шень А. Программирование: теоремы и задачи. – М.:МЦНМО, 1995. – 264 с.