

УСЛОВИЯ ЗАДАЧ ОСНОВНОГО ТУРА

Задача А. Деревья

Имя входного файла: `input.txt`
Имя выходного файла: `output.txt`
Ограничение по времени: 1 секунда
Ограничение по памяти: 16 мегабайт

Деревья имеют множество применений в информатике. Возможно, наиболее часто употребляемые деревья - бинарные деревья, но существуют и другие типы деревьев, которые также могут быть полезны. Один из примеров - упорядоченные деревья, в которых все потомки для любого заданного узла упорядочены. Количество потомков каждого узла произвольно и ограничений на это количество нет. Формально, упорядоченное дерево T состоит из ограниченного ряда узлов, таких, что:

- Существует один корневой узел, обозначенный как $r(T)$.
- Оставшиеся узлы разделены на упорядоченные деревья (поддеревья) T_1, T_2, \dots, T_m . Причем $r(T_1) < r(T_2) < \dots < r(T_m)$ и все $r(T_i)$ - это прямые потомки $r(T)$.
- Каждый упорядоченный набор $r(T_i)$ называется уровнем.

Часто удобнее представлять упорядоченное дерево в качестве бинарного дерева (так как в этом случае каждый узел будет занимать одинаковый объем памяти). Преобразование упорядоченного дерева в бинарное выполняется в следующем порядке:

1. Удалить все ребра от каждого узла к его потомкам.
2. Каждому узлу добавить ребро к его первому потомку в дереве (если такой есть) - как к левому потомку.
3. Каждому узлу добавить ребро к его следующему справа соседу по уровню (если такой есть) - как к правому потомку.

Пример преобразований:



В большинстве случаев, высота дерева (число ребер в самом длинном пути от корня к листу) после выполнения описанного преобразования возрастает.

Формат входного файла

Входной файл содержит одну строку, которая задает дерево методом *левого обхода в глубину*. Например, дерево, представленное выше, задается как `dudduduudu`, что означает 0 к 1, 1 к 0, 0 к 2 и т.д. Дерево имеет не менее 2 и не более 10000 узлов.

Формат выходного файла

Выходной файл должен содержать два натуральных числа, разделенных пробелом - высоту дерева до и после преобразования, рассмотренного выше.

Примеры

input.txt	output.txt
dudduduudu	2 4
ddddduuuuu	5 5

Задача В. Посмотри и скажи

Имя входного файла: `input.txt`
Имя выходного файла: `output.txt`
Ограничение по времени: 1 секунда
Ограничение по памяти: 16 мегабайт

Последовательность *посмотри и скажи* определяется следующим образом. Первый элемент последовательности - это произвольный набор цифр. Каждый последующий элемент определяется исходя из предыдущего элемента методом *вербального описания предыдущего элемента*. Например, ряд *122344111* может быть описан как *одна единица, две двойки, одна тройка, две четверки и три единицы*. Сам элемент представляет собой последовательность цифр, выделенную из вербального описания. Например, элемент, который следует после *122344111* в последовательности - это *1122132431*. Аналогично, ряд *101* следует после *1111111111*.

Ваша задача всего лишь найти следующий элемент по указанному текущему.

Формат входного файла

Входной файл состоит из одной строки длиной до 1000 цифр.

Формат выходного файла

Выходной файл должен содержать значение следующего элемента.

Примеры

<code>input.txt</code>	<code>output.txt</code>
122344111	1122132431
1111111111	101

Задача С. Перемещение сервера

Имя входного файла: `input.txt`
Имя выходного файла: `output.txt`
Ограничение по времени: 1 секунда
Ограничение по памяти: 16 мегабайт

У Михаила мощный сервер с сотнями параллельных процессоров и терабайтами оперативной памяти. Многие важные вычисления выполняются продолжительное время, поэтому электроэнергия должна подаваться к серверу непрерывно. По техническим причинам сервер Михаила должен быть перемещен из одного места серверной комнаты в другое. К счастью, сервер имеет два блока питания. Пока хотя бы один из двух блоков подключен к электрической розетке, сервер может продолжать работать. Когда сервер подключен к розетке, он может быть передвинут в любое положение, которое находится на расстоянии от розетки не большем, чем длина провода, используемого для подключения. Зная, к каким розеткам сервер подключен сначала и в конце, и расположение розеток в серверной комнате, Вы должны определить наименьшее количество переключений кабеля, которые нужно совершить, чтобы передвинуть сервер, сохраняя при этом его все время в рабочем состоянии. Переключением считается факт включения кабеля в розетку, в которую не включен второй кабель. В начальный момент оба кабеля подключены в начальную розетку, в конце - в конечную.

Формат входного файла

Первая строка входного файла содержит 5 чисел (разделенных пробелами): количество розеток в серверной комнате N ($2 \leq N \leq 1000$), номер начальной розетки (начиная с 1), номер конечной розетки (начиная с 1), длины двух электрических кабелей $D1$ и $D2$ с точностью до 3-х знаков после запятой ($0 < D1, D2 \leq 30000$).

За этим следуют N строк координат розеток (от 1-й до N -ой). Все координаты задаются двумя целыми числами x и y , разделенных пробелом, с наибольшим абсолютным значением 30000.

Координаты всех розеток различны. Начальная розетка и конечная розетка также различны.

Формат выходного файла

Выходной файл должен содержать одно число - минимальное число переключений, которое требуется, чтобы передвинуть сервер. Если это невозможно, выведите *Impossible*.

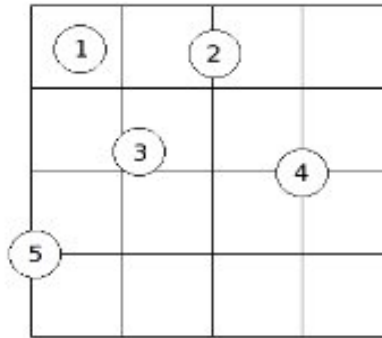
Примеры

input.txt	output.txt
4 1 4 2.000 1.000 0 0 0 4 4 0 4 4	Impossible
9 1 4 2.000 3.000 0 7 -6 2 -3 3 6 2 -6 -3 3 -3 6 -3 -3 -7 0 -7	8

Задача D. Бросание монетки

Имя входного файла: `input.txt`
Имя выходного файла: `output.txt`
Ограничение по времени: 1 секунда
Ограничение по памяти: 16 мегабайт

В одной популярной игре монетку бросают на стол с полем, которое покрыто квадратной сеткой. Призы распределяются по количеству клеток, покрытых монетой, когда она остановится: чем больше клеток она покрывает, тем лучше приз. На рисунке показаны результаты пяти бросков монеты:



В этом примере:

- монета 1 покрывает 1 клетку
- монета 2 покрывает 2 клетки
- монета 3 покрывает 3 клетки
- монета 4 покрывает 4 клетки
- монета 5 покрывает 2 клетки

Заметьте, что монетка может остановиться на границе игровой площадки (монета 5). Чтобы монета покрыла клетку, она должна покрыть часть клетки с положительной площадью. Другими словами, недостаточно просто коснуться границы клетки. Центр монеты может оказаться в любой точке игровой области с одинаковой вероятностью. Вы можете считать, что:

- 1) монета всегда останавливается, лежа на плоскости;
- 2) игрок достаточно хороший и может гарантировать, что центр монеты всегда оказывается на игровом поле (или на его границе).

Вероятность того, что монета закроет определенное количество клеток, зависит как от размеров клеток и монеты, так и от количества рядов и колонок клеток на игровом поле. В этой задаче от вас требуется написать программу, которая вычисляет вероятность того, что монета покроет определенное количество клеток.

Формат входного файла

Единственная строка входного файла содержит 4 целых числа m , n , t и c разделенных пробелами. Игровое поле состоит из m рядов и n колонок клеток, каждая из которых имеет сторону длиной t . Диаметр используемой монеты - c . $1 \leq m, n \leq 5000$ и $1 \leq c < t \leq 1000$.

Формат выходного файла

В выходной файл выведите 4 числа - вероятности того, что монета может покрыть 1 клетку, 2 клетки, 3 клетки и 4 клетки (в одну строку, разделяя числа пробелами). Вероятность следует выражать в процентном отношении, округленном до 4 знаков после десятичной точки.

Примеры

<code>input.txt</code>	<code>output.txt</code>
5 5 10 3	57.7600 36.4800 1.2361 4.5239
7 4 25 20	12.5714 46.2857 8.8293 32.3135

Задача Е. Аренда

Имя входного файла: `input.txt`
Имя выходного файла: `output.txt`
Ограничение по времени: 1 секунда
Ограничение по памяти: 16 мегабайт

Курорт *Пятый сезон* состоит из ряда домов, которые довольно часто занимают их основные владельцы. Однако, в остальное время, они выполняют функцию домов под аренду. Поскольку на курорте не больше 26 домов, они идентифицируются прописными латинскими буквами.

Однажды звонит телефон менеджера курорта. Она принимает запрос на аренду дома на отпуск с датой прибытия 2 декабря и датой отъезда 9 декабря. Она смотрит на таблицу заказов, но не находит свободный дом на указанный период. Большая часть существующих заказов была сделана владельцами соответствующих домов (тех, кто хочет остаться в своих собственных помещениях), поэтому нежелательно перемещать существующую бронь из одного помещения в другое. Продолжая рассматривать таблицу заказов, менеджер, однако, находит выход и говорит: *Я могу поселить Вас в здание В на первые 3 ночи, а потом переселить в здание F на остальное время отдыха. Вас это устроит?* Человек соглашается, заказ готов. Заметьте, что заказы делаются на ночь, так что заказ на одну ночь означает, что гость уедет на следующий день после прибытия.

Ваша задача - принять новый заказ на бронирование (не трогая существующие) с минимальным количеством переселений из дома в дом.

Формат входного файла

В первой строке входного файла находятся 2 положительных целых числа M и N . M - количество последовательных дней, на которые менеджер курорта ведет таблицу заказов, а N - количество домов на курорте. Помещения обозначаются заглавными латинскими буквами, начиная с A . Всего может быть не более 100 дней и не менее 3 помещений в таблице заказов. Дни обозначаются 1,2,3 ... M .

В следующих M строках находится таблица заказов. Каждая строка таблицы относится к определенному дню (в порядке 1, 2, 3, и т.д.), а каждый столбец таблицы - к определенному помещению курорта (в порядке A , B , C и т.д.) Символ X (заглавная буква *икс*) означает, что рассматриваемое помещение зарезервировано на этот день, тогда как O (заглавная буква *о*) означает, что помещение свободно.

За таблицей заказов следует строка запроса на аренду дома, состоящая из двух целых величин: даты заезда и даты отъезда. Дата заезда находится в пределах от 1 до M . Дата отъезда больше даты заезда и меньше или равна $M+1$.

Формат выходного файла

Если запрос на аренду может быть удовлетворен, то выходной файл должен содержать расписание (график) заказа с минимальным количеством переносов (из одного помещения в другое) на протяжении пребывания на курорте. Каждая строка расписания соответствует заселению в одно помещение и выглядит следующим образом: *<помещение>: <начальная дата> - <конечная дата>*, где *<помещение>* - это помещение, *<начальная дата>* - дата заезда гостя в помещение, а *<конечная дата>* - дата выезда гостя из помещения. Строки в расписании должны быть расположены в возрастающем порядке с даты заезда.

При наличии нескольких графиков с минимальным количеством переносов выберите график, который использует наименьшее *наименование помещения* в первый день (так, помещение A имеет приоритет к помещению B). Если это сделать невозможно, выберите график с наименьшим номером помещения во второй день и так далее.

Если запрос не может быть удовлетворен, напечатайте строку *Not available* вместо расписания.

Примеры

input.txt	output.txt
10 7 XXXXXXX XOXXXXO XOXXXXO XOXXXXO OXXOXOX XOXOXOX OXXOXOX OXXXXOX XXXXXXX XXXXXXX 2 9	B: 2-5 F: 5-9

Задача F. Бейсбол

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	1 секунда
Ограничение по памяти:	16 мегабайт

Бейсбол состоит из 9 иннингов. Если после 9 иннингов игра оказывается вничью, играется дополнительный иннинг, пока счет не оказывается в чью-либо пользу в конце этого иннинга. Каждый иннинг разделен на 2 половины, во время которых одна команда атакует, другая команда - защищается. Команда-гость атакует в первой половине, тогда как хозяева - во вторую половину. Если команда хозяев ведет в счете в начале второй половины 9-го иннинга, игра останавливается, т.к. победитель уже определен.

Каждая команда придерживается порядка битья, при котором 9 игроков команды бьют на протяжении атаки половину иннинга. В каждой половине иннинга игроки атакующей команды бьют в соответствии с порядком битья. Первый бьющий в первом иннинге является первым в порядке битья. Каждый последующий бьющий является следующим игроком в порядке битья. Если предыдущий бьющий - это последний игрок в порядке битья, следующий бьющий - снова первый игрок в порядке битья. В каждом последующем иннинге первый игрок, который должен бить - это игрок, следующий за последним игроком, который бил в порядке битья.

Если игрок совершил удачный удар, то игрок с битой бежит к первой базе, а игроки, уже находящиеся на базе, бегут на одну базу вперед (в настоящих играх игрок может бежать сразу на несколько баз, но мы не будем рассматривать этот случай в данной задаче). Игрок должен достигнуть первой базы, второй базы, третьей базы и в конце концов - домашней базы, чтобы ему засчитали пробежку. Когда игрок достигает домашней базы, его возвращают на скамью, где он ждет своей следующей возможности ударить. Половина иннинга продолжается сколько угодно времени, пока 3 атакующих игрока не ошибутся 3 раза при ударе (каждый неудачный удар - это *аут*). Когда это происходит, игроки, оставшиеся на первой, второй и третьей базах, возвращаются на скамью, и пробежка им не засчитывается. В конце игры побеждает команда, которой засчитано наибольшее количество пробежек.

В определенных ситуациях игрок может *принести себя в жертву*, чтобы дать преимущество игрокам, уже находящимся на базах. Если жертва удачна, бьющий игрок уходит на скамью, но каждый, находящийся на базе, бежит до следующей базы. Игрок, добегающий до домашней базы таким образом получает пробежку, если только бьющий не был третьим, пропустившим мяч в иннинге. В последнем случае половина иннинга заканчивается и пробежка не засчитывается. Если пропуск неудачен, бьющий оказывается в ауте, и никто из игроков не достигает успеха. Бьющий может совершить попытку пропустить удар всякий раз когда на второй базе стоит игрок без аутов или когда на третьей базе находится игрок максимум с одним аутом.

Одним из наиболее часто используемых видов статистики является *процент попаданий* (число между 0 и 1) для каждого игрока, показывающий вероятность совершения игроком удачного удара. Аналогично, *процент уступок* игрока - это вероятность совершения удачной уступки.

В данной задаче вам дан процент ударов и процент уступок каждого из 9 игроков в команде, а также порядок битья. Ваша задача симулировать игру в бейсбол. В имитации требуются случайные числа, для их получения необходимо использовать следующий генератор случайных чисел:

$$X(n+1) = (X(n) * 25173 + 13849) \text{ div } 65536$$

где $X(n)$ - предыдущее случайное число, а $X(n+1)$ - следующее случайное число. Генератор должен работать с 32-х битными беззнаковыми целыми числами. Начальное значение $X(0) = 1$; первое случайное число, сгенерированное вашей программой - $X(1)$.

На каждом шаге имитации необходимо выяснить были ли удар или уступка успешными, а потом сгенерировать следующее случайное число. Удар (или уступка) успешны, если: (случайное число / 65536) <= процента попаданий (уступок). Деление в данной формуле является делением с плавающей запятой.

Формат входного файла

Входной файл включает порядок биття команды гостей и следующий за ним порядок биття домашней команды. Порядок биття каждой команды начинается со строки, в которой записано имя команды (максимум 15 символов без пробелов). В следующих 9 строках записаны 9 игроков, перечисленных в порядке, в котором они делают удар. Каждая из этих строк включает имя игрока (максимум 15 символов без пробелов), пробел, затем - число с плавающей запятой (до 3 знаков после десятичной точки), обозначающее процент его ударов, пробел, и наконец процент его уступок (до 3 знаков после десятичной точки). Ведущий 0 в числах с плавающей точкой может быть опущен. Процент попаданий находится между .200 и .400, а процент уступок - между .300 и .750. Ни одна игра не будет включать больше, чем 200 иннингов.

Формат выходного файла

Выходной файл должен содержать имена игроков, которые зарабатывают попадания и пробежки отдельно для каждого иннинга, в порядке, в котором они происходят в игре. Используйте формат вывода, как в примере выходных данных. Каждая строка описывает одно попадание или пробежку. Попадание начинается со слова *Hit*, пробежка - *Run*. Далее в скобках указывается номер иннинга, затем (после двоеточия и пробела) - Имя игрока и название команды. Для каждого иннинга сначала указываются попадания, а потом пробежки.

В конце игры выведите количество пробежек и хитов, засчитанных каждой команде, начиная с команды гостей.

Примеры

input.txt	output.txt
Rangers	Hit(1): Hill BlueJays
Young .213 .523	Hit(1): Overbay BlueJays
Kinsler .207 .602	Hit(1): Adams BlueJays
Sosa .254 .300	Hit(3): Catalanotto Rangers
Laird .220 .432	Hit(3): Wells BlueJays
Byrd .206 .749	Hit(3): Hill BlueJays
Wilkerson .236 .508	Hit(3): Adams BlueJays
Catalanotto .272 .483	Run(3): Wells BlueJays
Teixeira .297 .573	Hit(4): Kinsler Rangers
Saltalamacchia .243 .632	Hit(5): Catalanotto Rangers
BlueJays	Hit(6): Sosa Rangers
Wells .378 .502	Hit(6): Laird Rangers
Hill .276 .544	Hit(6): Rios BlueJays
Overbay .372 .694	Hit(7): Wilkerson Rangers
McDonald .373 .618	Hit(7): Teixeira Rangers
Adams .320 .690	Hit(7): Stairs-BlueJays
Rios .300 .450	Rangers: 0 runs, 7 hits
Johnson .379 .559	BlueJays: 1 runs, 8 hits
Stairs .302 .621	
Zaun .346 .515	

Задача G. Работа в команде

Имя входного файла: `input.txt`
Имя выходного файла: `output.txt`
Ограничение по времени: 1 секунда
Ограничение по памяти: 16 мегабайт

Тренер после конкурса по программированию расстроен недостатком командной работы в своих командах. Он решил продемонстрировать своим студентам важность командной работы, используя известную аналогию: легко взять отдельную веточку и сломать ее на две половинки. Но если вы соберете 3 веточки вместе (т.е. 3 участника команды), вам потребуется приложить значительно большее усилие, чтобы сломать веточки. Тренер уверен, что программисты поймут важность работы в команде после его демонстрации.

Итак, тренер пришел в лес, чтобы собрать веточки. Он знает, что демонстрацию всегда нужно отрепетировать, чтобы избежать проблем. Он уверен, что практически не возможно сломать три ветки, собранных вместе и что очень легко сломать отдельные ветки. Но нет! В процессе эксперимента каждая из собранных им веток переломилась на несколько частей! Тренер может выйти и собрать еще веток, но как он узнает, что ветки поведут себя так, как нужно, на демонстрации?

И тут тренеру приходит в голову отличная идея: просто склеить кусочки вместе, чтобы собрать ветки побольше! Оказалось, что каждый кусочек можно спокойно склеить с любым другим, даже если эти два кусочка были частями разных веточек. То есть, он может восстановить ветки, склеивая вместе 2 и более кусочка. Теперь каждая отдельная ветка может быть легко сломана в точке соединения. Однако если соединения 2-х веток находятся рядом (при сборке их в пакет из 3-х веточек), их будет так же легко сломать собранные вместе. Значит, нужно *реконструировать* ветки таким образом, чтобы никакие точки соединения не соприкасались. Более того, желательно, чтобы собранные ветки были настолько длинными, насколько это возможно. Наконец, три собранные веточки должны быть одинаковой длины: тренер не хочет быть понятым так, что один участник команды лучше другого. Возможно, что некоторые кусочки останутся неиспользованными. Естественно, каждый кусочек может быть использован лишь 1 раз.

Формат входного файла

Входной файл состоит из одной строки. Первое число обозначает количество кусочков (N). Каждое из последующих чисел N является положительным целым числом, обозначающим длину каждого кусочка. Максимум может быть 13 кусочков, а длина каждого из них не более 25.

Формат выходного файла

Выходной файл должен содержать одно число - максимально возможную длину каждой из трех собранных веток. Если невозможно собрать заново 3 ветки, удовлетворяющие условиям, то выведите 0.

Примеры

<code>input.txt</code>	<code>output.txt</code>
10 4 2 3 7 8 9 1 2 3 4	14
10 1 2 3 4 5 6 7 8 9 10	18

Задача Н. Преобразования звука

Имя входного файла: `input.txt`
Имя выходного файла: `output.txt`
Ограничение по времени: 1 секунда
Ограничение по памяти: 16 мегабайт

Трансформация дискретных волн - распространенный инструмент для компрессии звукового сигнала. В этой задаче ваша цель - написать программу, которая будет распаковывать сигнал (список целых чисел), который сжат простой волновой трансформацией.

Чтобы понять, как работает простая волновая трансформация, предположим, что у нас есть список целых чисел. Мы вычисляем сумму и разность каждой пары последовательно идущих чисел, занося результаты в 2 списка сумм и разностей, каждый из которых имеет длину в половину длины начального списка.

Формально:

оригинальный список - $a(1), a(2), \dots, a(n)$.

i -тая сумма $s(i)$ и разность $d(i)$ вычисляются следующим образом: $s(i) = a(2*i-1) + a(2*i)$, $d(i) = a(2*i-1) - a(2*i)$ для i от 1 до $n/2$.

Затем идет перестроение - вначале записывается набор сумм, а затем - набор разностей.

Например, если исходный сигнал имеет вид:

5, 2, 3, 2, 5, 7, 9, 6, то сигнал после преобразования будет иметь вид:

7, 5, 12, 15, 3, 1, -2, 3.

Затем этот же процесс применяется к первой половине трансформированного сигнала, и далее рекурсивно, пока длина сигнала вывода не станет равна 1. В нашем примере окончательно трансформированный сигнал такой:

39, -15, 2, -3, 3, 1, -2, 3.

Длина начального списка кратна степени 2, а каждое число в сигнале - это целое число от 0 до 255 (включительно).

Формат входного файла

Входной файл состоит из одной строки, которая начинается с целого числа N ($1 \leq N \leq 256$) - длины сигнала. Следующие N целых чисел - это значения закодированного сигнала.

Формат выходного файла

Выходной файл должен содержать раскодированный сигнал (значения разделяются пробелами).

Примеры

<code>input.txt</code>	<code>output.txt</code>
8 39 -15 2 -3 3 1 -2 3	5 2 3 2 5 7 9 6
4 10 -4 -1 -1	1 2 3 4

Задача I. Простое сложение

Имя входного файла: `input.txt`
Имя выходного файла: `output.txt`
Ограничение по времени: 1 секунда
Ограничение по памяти: 16 мегабайт

В современном мире студенты слишком много полагаются на калькуляторы и компьютеры, производя простейшие вычисления. К сожалению, нередко можно увидеть университетских студентов, которые не могут выполнить простой арифметический расчет без электронных приборов. Профессор Пеано видел таких предостаточно. Он решил взять дело в собственные руки и заставить своих студентов выучиться базовым арифметическим навыкам: хотябы сложению неотрицательных целых чисел. Для этого он решил вернуться к основам и представить неотрицательные целые числа следующим образом:

0 представлен пустым множеством $\{\}$.

Для любого числа $n > 0$, оно представлено множеством, включающим в себя представления всех неотрицательных целых чисел, меньших чем n .

Например, первые 4 неотрицательных целых числа представлены следующим образом:

0 - $\{\}$

1 - $\{\{\}\}$

2 - $\{\{\}, \{\{\}\}\}$

3 - $\{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\}$

и так далее. Размер множества - это в точности то число, которое оно представляет. Хотя элементы множества могут быть и неупорядочены, профессор Пеано требует, чтобы все элементы были расположены в порядке возрастания их размеров. Профессор считает, что не существует калькуляторов и компьютерных программ, которые могут работать с числами, записанными в такой нотации.

Не удивительно, что многие студенты не могут справиться с этим заданием и завалят курс, если не получат помощь. И помочь им - Ваша задача! Вам нужно написать программу, чтобы помочь студентам решать примеры на сложение.

Формат входного файла

Входной файл содержит 2 строки, каждая из которых содержит неотрицательное целое число, представленное в описанной нотации. На каждой строке есть только символы $\{$, $\}$ и *запятая*. Сумма двух заданных целых чисел будет не больше 15.

Формат выходного файла

Выходной файл должен содержать одну строку - сумму чисел в указанной нотации.

Примеры

<code>input.txt</code>	<code>output.txt</code>
$\{\}$ $\{\}$	$\{\}$
$\{\{\}\}$ $\{\{\}, \{\{\}\}\}$	$\{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\}$