Министерство образования и науки Украины

Зимняя школа по программированию

Харьков, ХНУРЭ
2009

Оглавление

День первый. Контест Виталия Неспирного и Антона Лунёва 5
Об авторах... 5
Теоретический материал. Близость точек на плоскости 6
Задачи и разборы 13
Задача A. Выборы вождя 13
Задача B. Покер 15
Задача C. Каждый третий бесплатно 17
Задача D. Распределение 19
Задача E. Кладбище 21
Задача F. Продовольственная программа 23
Задача G. Призыв в армию 30
Задача H. Родственники 32
Задача I. Деление земель 37
Задача J. Экстремальные расстояния 40
Задача K. Наименьший квадрат 41
Задача L. Вражда 43
Задача M. Дирихле и Фибоначчи 46
Задача N. Похожие слова 47
Задача O. Экстремальный граф 49
Задача P. Попарные суммы 50
Задача Q. Биномиальные коэффициенты 53
Задача R. Биномиальные коэффициенты 2 55
Задача S. Пирамида из чисел 58
Задача T. k-суммы с повторениями 59
День Второй. Контест Михаила Дворкина 62
Об авторе... 62
Теоретический материал. Конечные автоматы и регулярные выражения 62
Задачи и разборы 71
Задача A. Минимизация ДКА 71
Задача B. Обращение ДКА 72
Задача C. ДКА, проверяющий делимость 73
Задача D. Укладка плиток 74
Задача E. Непоглощающий ДКА 76
Задача F. Распознавание конечного языка 77
Задача G. Регулярное выражение 78
Задача H. Автомат-собеседник для Эллочки-людоедки 80
Задача I. Минимизация ДКА 81
Задача J. Обращение ДКА 83
Задача K. ДКА, проверяющий делимость 85

Задача L. Укладка плиток 87
Задача M. Непоглощающий ДКА 90
Задача N. Распознавание конечного языка 92
Задача O. Регулярное выражение 94
Задача P. Автомат-собеседник для Эллочки-людоедки 96
День третий. Контест Виталия Виталия Гольдштейна 100
Об авторе... 100
Теоретический материал. Потоки в сетях 100
Задачи и разборы 103
Задача A. Дартс 103
Задача B. Сочинитель стихов 105
Задача C. Число инверсий 107
Задача D. Дириктивы include 108
Задача E. Простецкие числа 110
Задача F. Неточный поиск 111
Задача G. Такси 113
Задача H. Великая стена 115
Задача I. Разрез 116
Задача J. Испорченный паркет 117
День четвертый (21.02.2009 г.) Контест Виталия Гольдштейна 119
Теоретический материал. Близость точек на плоскости 119
Задачи и разборы 120
Задача A. Мороженое 120
Задача B. Коллекционирование монет 122
Задача C. Уголки 124
Задача D. Встроенная очередь команд 125
Задача E. Клуб “Двоичный кот” 128
Задача F. Балансировка нагрузки 130
Задача G. Охлаждение реактора 131
Задача H. Пешеходные зоны против кольцевых 133
Задача I. Кони 134
Задача J. Сложите многоугольник 136
День пятый (22.02.2009 г.) Контест Михаила Медведева 138
Об авторе... 138
Задачи и разборы 173
Задача A. Одинаковые шары 173
Задача B. Дни рождения 175
Задача C. Толстая монета 177
Задача D. Дуэлянты 179
Задача E. Лес 182
Задача F. Счастливые пятерки 185

Задача G. Произведение матриц 187
Задача H. Треугольники в многоугольнике 188
Задача I. Обработка текста 191
Задача J. Чемпионат мира 193
День Теодора Заркуа и его учеников 197
Об авторе... 197
Теоретический материал. 198
Задачи и разборы 204
Задача A. Максимальное число 204
Задача B. Двоичные строки 206
Задача C. Стрекоза и муравей 207
Задача D. Кладоискатели 210
Задача E. Однажды в Китае 212
Задача F. Возвращение Супермена 214
Задача G. Нелюбимые цифры 217
Задача H. Шоколад 220
Задача I. Похищение невесты 221
Задача J. Сегменты 224
Задача K. Шоколадная Фабрика 226
Задача L. Строки 227
Задача M. Стрекоза и муравей 2 230
Задача N. Нелюбимые цифры 2 232
Задача O. Снова палиндромы 234
Задача P. Преобразование 236
Задача Q. Равносильность 237
Задача R. Однажды в Китае 2 239

День первый (18.02.2009 г.)
Контест Виталия Неспирного и Антона Лунёва

Об авторах...

Неспирный Виталий Николаевич, кандидат физико-математических наук, младший научный сотрудник отдела технической механики Института прикладной математики и механики НАН Украины, ассистент кафедры теории упругости и вычислительной математики Донецкого национального университета, заместитель председателя жюри II-III этапов Всеукраинской олимпиады школьников по ин-форматике в Донецкой области, координатор Донецкого сектора Открытого кубка по программированию, тренер команд математического факультета Донецкого национального университета.

Основные достижения:

- Занял 1-е место на Всеукраинской студенческой олимпиады по информатике (Николаев, 2000).
- В составе команды математического факультета ДонНУ занял 1-е место на Всеукраинской ACM-олимпиаде по программированию и 8-е в личном первенстве (Винница, 2001).
- Подготовил призеров международных школьных олимпиад - П.Луференко (2002) и Р.Ризванов (2006).

Лунёв Антон Андреевич, студент 5-го курса специальности математика Донецкого национального университета. Занимается математикой. Основные научные интересы:

- точные константы в неравенствах для промежуточных производных;
- корневые решения обобщенного уравнения Маркова;
- полнота и базисность системы корневых векторов дифференциальных операторов. Увлекается программированием.

Основные достижения:

- Трижды (2005, 2006, 2007) был награжден первым призом на международной математической олимпиаде для студентов IMC.
- В 2006, 2007 годах награжден серебряной медалью, а в 2008 - золотой, на международной студенческой математической олимпиаде в Иране.
- В 2008 году занял в составе команды Донецкого национального университета 7-е место из 44 участвующих команд на региональной олимпиаде ACM по программированию (Southeastern European Regional Contest).
- Занял 9-е место и награжден дипломом третьей степени по итогам зимней серии индивидуальных соревнований по программированию SnarkNews Winter Series 2009.

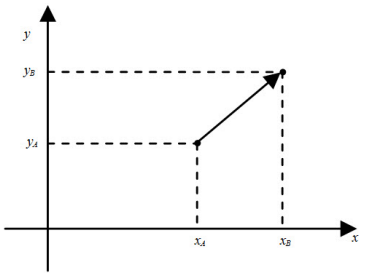
Теоретический материал. Близость точек на плоскости

1. Инструменты для решения геометрических задач: точки, векторы на плоскости и операции над ними.

Введем на плоскости декартову систему координат Oxy. Тогда каждая точка будет задаваться двумя координатами (x, y). Если будут заданы две

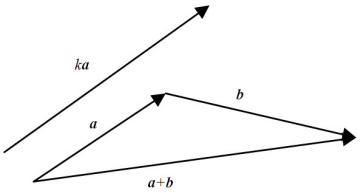
точки, то их можно соединить отрезком. Направленный отрезок, соединяющий две точки A и B (причем A считается началом вектора) называется вектором – одна из этих точек определяет. При этом отождествляем параллельные векторы одинаковой длины, направленные в одну сторону (то есть, рассматриваем свободные векторы).

Вектор на плоскости задается также двумя координатами. Пусть точка $A(x_A, y_A)$ – начало вектора, а $B(x_B, y_B)$ – конец вектора. Тогда вектор $\boldsymbol{a}$=AB будет иметь координаты $(x_B - x_A, y_B - y_A)$.



Два вектора $\boldsymbol{a}(a_x, a_y)$ и $\boldsymbol{b}(b_x, b_y)$ можно сложить. При сложении двух этих вектор, нужно отложить от конца вектора $\boldsymbol{a}$ вектор $\boldsymbol{b}$. Тогда суммой будет называться вектор $\boldsymbol{a}$+$\boldsymbol{b}$, соединяющий начало вектора $\boldsymbol{a}$ с концом вектора $\boldsymbol{b}$. Его координаты выразятся через исходные, как $(a_x + b_x, a_y + b_y)$.

При умножении вектора $\boldsymbol{a}$ на некоторую константу k, его длина увеличится в $|k|$ раз. При этом направление останется тем же (при положительном k) или изменится на противоположное (при отрицательном k).



Длина вектора $\boldsymbol{a}$ (или его модуль $|\boldsymbol{a}|$) может быть вычислена как гипотенуза прямоугольного треугольника и равна $\sqrt{a_x^2 + a_y^2}$.

Наиболее важными операциями для векторов является их произведения. Благодаря простым формулам для вычисления этих величин и их геометрическому смыслу, они часто используются для проверки взаимной ориентации некоторых векторов относительно друг друга.

Скалярное произведение векторов $\boldsymbol{a}(a_x, a_y)$ и $\boldsymbol{b}(b_x, b_y)$ вычисляется по формуле $\boldsymbol{a} \cdot \boldsymbol{b} = a_x \cdot b_x + a_y \cdot b_y$. С другой стороны имеем выражение для скалярного произведения через геометрические величины (длины векторов и угол между ними) – $\boldsymbol{a} \cdot \boldsymbol{b} = |\boldsymbol{a}| \cdot |\boldsymbol{b}| \cos \varphi$, где φ – это угол между векторами $\boldsymbol{a}$ и $\boldsymbol{b}$. Поскольку косинус – четная функция, то не имеет значения знак этого угла – он может отмеряться как по часовой стрелке, так и против. Знак скалярного произведения позволяет нам проверить является ли угол между некоторыми направлениями прямым, острым или тупым. Величина же скалярного произведения равна алгебраической величине проекции одного из этих векторов на направление, задаваемого другим, умноженной на модуль второго вектор.

Векторное произведение двух векторов $\boldsymbol{a}(a_x, a_y)$ и $\boldsymbol{b}(b_x, b_y)$ – это величина, вычисляемая по формуле $\boldsymbol{a} \times \boldsymbol{b} = a_x \cdot b_y - a_y \cdot b_x$. С другой стороны эта

величина связана с геометрическими характеристиками данных векторов формулой $\boldsymbol{a} \times \boldsymbol{b} = |\boldsymbol{a}| \cdot |\boldsymbol{b}| \sin \varphi$. Здесь уже важно, что угол отмеряется от $\boldsymbol{a}$ до $\boldsymbol{b}$, и считается положительным, если направление вращения по данному углу от $\boldsymbol{a}$ до $\boldsymbol{b}$ будет против часовой стрелки. Таким образом, векторное произведение в отличие от скалярного является кососимметричной операцией (меняет знак при изменении порядка операндов). Кроме того, по абсолютной величине векторное произведение равно удвоенной площади треугольника, стороны которого определяются данными векторами.

Отсюда в частности следует, что площадь треугольника с целочисленными координатами является целой или полуцелой (дробная часть равна 0.5). Кроме того, векторное произведение равно нулю тогда и только тогда, когда векторы коллинеарны.

(Вообще говоря, в аналитической геометрии векторное произведение двух векторов вводится как вектор, перпендикулярный плоскости, содержащей данные векторы, с конца которого вращение от первого ко второму на меньший угол происходит против часовой стрелки, а величина равна площади треугольника, натянутому на них. То, что мы здесь приняли за векторное произведения – это скаляр, равный третьей координате векторного произведения двух данных векторов, если их расположить на координатной плоскости Oxy трехмерного пространства).

2. Выпуклая оболочка.

Выпуклым называется такое множество точек плоскости, которое обладает тем свойством, что для любых двух точек, принадлежащих множеству, отрезок их соединяющий, целиком содержится в данном множестве.

Если же исходное множество не является выпуклым, то можно построить множество, содержащее исходное в качестве подмножества. Оказывается, что среди всех таких выпуклых множеств существует одно минимальное (в том смысле, что содержится в любом другом выпуклом множестве, содержащем исходное). Оно и называется выпуклой оболочкой исходного множества.

Если задано конечное множество точек, то, как нетрудно увидеть, его выпуклой оболочкой будет многоугольник, вершинами которого будут точки исходного множества. Таким образом, задача построения выпуклой оболочки заключается в выделении таких точек и перечислении их в порядке обхода по контуру (например, против часовой стрелки).

Для решения этой задачи могут быть использованы различные алгоритмы. Рассмотрим два их них.

Алгоритм Джарвиса. Найдем сначала какую-либо точку, которая наверняка принадлежит выпуклой оболочке. Такой будет, например, точка A_1 наименьшая в лексикографическом порядке. Она будет первой вершиной выпуклой оболочки. Теперь нам нужно найти следующую за ней точку в выпуклой оболочке, следующую за вершиной A_1. Эту точку A_2 можно найти как имеющую наименьший полярный угол, относительно точки A_1 как начала координат с направлением, выбранным в отрицательную сторону по оси Oy. При наличии нескольких таких точек следует выбирать наиболее удаленную от точки A_1. Аналогично, точка A_3 будет иметь наименьший полярный угол относительно с началом координат в точке A_2 с направлением выбранной оси по вектору A_1A_2. Продолжая последовательно находить следующие точки аналогичным образом, мы в конечном итоге вернемся в исходную точку A_1. Многоугольник замкнется и тем самым будет построена полностью выпуклая оболочка данного множества. При нахождении очередной точки (как как имеющей минимальный угол) нам может помочь векторное произведение. Точка X будет предшествовать другой Y (иметь меньший полярный угол) тогда и только тогда, когда векторное произведение A_iX на A_iY будет положительным или нулевым (коллинеарные векторы), но при этом длина A_iX будет больше чем длина A_iY, где A_i – последняя найденная точка выпуклой оболочки. Благодаря тому, что все точки будут находиться в одной полуплоскости относительно вектора $A_{i-1}A_i$, указанное отношение будет являться отношением порядка.

Алгоритм будет иметь сложность O(Nh), где h – это количество точек, принадлежащих выпуклой оболочке. В худшем случае эта величина может достигать значения N (когда все точки попадают на выпуклую оболочку), и значит при таких “плохих” исходных данных время работы алгоритма Джарвиса может достигать $O(N^2)$.

Алгоритм Грехема. Возьмем произвольную точку O. Есть два варианта – эта точка может оказаться внутри выпуклой оболочки, или вне ее. Предположим, что точка O попадет внутрь выпуклой оболочки. Тогда мы можем провести лучи из нее на все точки заданного множества и отсортировать их по полярному углу относительно какой-либо оси, проходящей через O. Запишем эти точки в некоторый циклический список. Выберем три последовательные точки этого списка (p_1 – предыдующая точка, p_2 – текущая точка, p_3 – предыдующая точка). Если эти три точки образую левый поворот (вращение от вектора p_1p_2 к вектора p_2p_3 происходит против часовой стрелки), то (пока) нет оснований удалять текущую вершину p_2, и мы переходим от текущей к следующей вершине (делаем ее текущей и соответствующим образом перемещаем указатели p_1 и p_3. Если же $p_1p_2p_3$

образуют правый поворот, то мы должны удалить вершину p_2 и вернуться к предыдущей вершине p_1 (нужно будет проверить и ее). Так будем продолжать до тех пор, пока мы не просмотрим все вершины. После всех удалений в нашем списке останутся лишь вершины выпуклой оболочки.

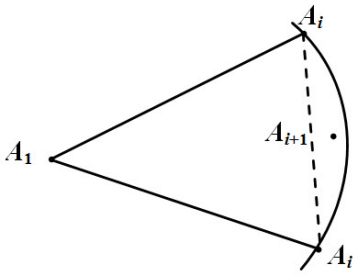
Нетрудно убедиться в том, что сам обход вершин по Грехему потребует не более O(N) операций (действительно, на каждом шаге мы либо переходим к следующей вершине, либо удаляем текущую: продвинуться вперед мы можем не более N раз, и удалить мы сможем максимум N вершин). Поэтому сложность всего алгоритма будет определяться сложностью сортировки – O(N log N).

Если же точка O будет находиться за пределами выпуклой оболочки, мы сразу сможем найти две точки заведомо принадлежащие выпуклой оболочке. Это точки A_{min} и A_{max}, полярный угол которых относительно O, будет минимальным и максимальным соответственно. Выпуклая оболочка тогда будет разбита на две цепочки – ближняя и дальняя относительно точки O. Дальняя цепочка может быть получена обходом Грехема от A_{min} до A_{max}, а ближняя – обходом Грехема от A_{max} до A_{min}.

Удобнее всего использовать точку, удаленную на бесконечное расстояние в направлении оси Oy. Тогда роль лучей будут выполнять прямые параллельные оси Oy, а роль углов – координаты x исходных точек.

3.Наиболее удаленные точки.

Предположим, что нам заданы N точек, и требуется найти пару точек, расстояние между которыми будет максимальным. Задача кажется на первый взгляд чрезвычайно просто. Ведь достаточно проверить $N(N\text{-}1)/2$ пар точек и выбрать максимум из расстояний между ними. Это расстояние называется диаметром данного множества. Однако оказывается, что указанное решение, несмотря на очевидную правильность, является далеко не оптимальным.



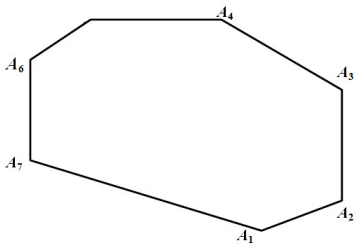
Первое замечание, которое может помочь в решения задачи – это то, что диаметр множества равен диаметру его выпуклой оболочки. Тогда мы найдем выпуклую оболочку исходного множества точек и уже на ней находить наиболее удаленные точки. Может оказаться, что наше множество будет существенно “прорежено” выпуклой оболочкой, но вполне вероятен и другой вариант – что на выпуклой оболочке останутся все точки. Так что единственное, что мы сделали этим шагом – упорядочили их таким образом, чтобы они задавали обход многоугольника.

На первый взгляд может показаться, что если мы зафиксируем первую точку и будем в порядке обхода по контуру переходить к последующим

точкам, то у нас будет сначала монотонно увеличиваться расстояние, а затем уменьшаться. Но это предположение неверно в общем случае. И мы можем построить соответствующий пример.

Зафиксируем точку A_1. Допустим, у нас есть две точки A_i и A_{i+2} находящиеся на одинаковом расстоянии R от точки A_1. Нетрудно видеть, что можно между A_i и A_{i+2} найти такую точку A_{i+1}, которая бы могла принадлежать выпуклой оболочке, но при этом расстояние до нее было бы меньше R. (В качестве такой точки можно выбрать любую точку сегмента круга, расположенного между хордой A_iA_{i+2} и дугой A_iA_{i+2}. Ясно, что небольшие сдвиги могут сделать максимально удаленной от A_1 любую из точек A_i и A_{i+2}. Причем, может быть такое, что таких "локально экстремальных точек будет довольно много, и между ними будет находиться гораздо больше точек.

Ключом к решению задачи является следующее наблюдение: диаметр выпуклой фигуры равен наибольшему из расстояний между двумя параллельными опорными прямыми (то есть такие, относительно которых все множество лежит в одной полуплоскости) к этой фигуре.



Очевидно, что не через каждую пару точек можно провести параллельные опорные прямые. Например, на рисунке никакие две опорные прямые проходящие через вершины A_4 и A_6 не будут параллельными.

Назовем пара точек, через которые можно провести параллельные опорные прямые, противолежащей парой. В силу сделанного нами наблюдения, нам следует искать диаметр, перебирая не все пары точек, а лишь противолежащие.

Оказывается, что количество противолежащих пар сравнимо с N. И для их получения может быть использован следующий алгоритм. Найдем первую (в порядке обхода по контуру) точку A_i, которая будет образовывать противолежащую пару с A_1. Эта точка может быть найдена как максимально удаленная от отрезка A_NA_1. Вот это расстояние (от фиксированного отрезка) уже будет обладать той монотонностью, которую мы ожидали для расстояния от фиксированной точки до остальных. Все точки, которые расположены между A_i и точкой A_j, удаленной на максимальное расстояние уже от отрезка A_1A_2, включая и сами точки A_i и A_j, будут образовывать противолежащие пары с A_1. Далее перейдем к вершине A_2 – цепочка противолежащих к ней вершин будет начинаться уже в вершине A_j, а заканчиваться в вершине, наиболее удаленной от отрезка A_2A_3. Продолжая продвигаться по контуру, мы можем закончить обход тогда, когда

противолежащая к текущей вершина достигнет уже точки A_1 (второй раз брать те же пары, но с обратным порядком не имеет смысла).

Таким образом, после получения выпуклой оболочки (за O(N log N)) получение всех противолежащих вершин (а значит и диаметра) потребует O(N) операций.

Замечание. При получении противоположных пар нужно аккуратно разобрать случай, когда у нас есть параллельные ребра, но для нахождения диаметра и наиболее удаленных точек это не имеет значения.

4. Пара ближайших точек.

По прежнему, нам заданы N точек, и требуется найти пару точек, расстояние между которыми будет минимальным. Несмотря на внешнюю похожесть на предыдущую задачу, ее эффективное решение предполагает совсем другой подход.

Применим метод типа "разделяй и властвуй". Разобьем исходное множество S некоторой прямой на два подмножества S_1 и S_2 с примерно одинаковым количеством точек. Затем рекурсивно найдем минимальные расстояния δ_1 и δ_2 для каждого подмножества $S1$ и $S2$. Ясно, что расстояние между ближайшей парой точек в множестве S не превосходит $\delta = min(\delta_1, \delta_2)$. Но может быть еще меньше за счет пар точек одна, из которых принадлежит множеству $S1$, а другая – S_2. Ясно, что имеет смысл рассматривать лишь точки, которые находятся на расстоянии не более δ от разделяющей прямой – тем самым мы рассматриваем точки, лежащие в полосе ширины 2δ. Отсортируем в точки в каждой полуполосе ширины δ все точки этих множеств по проекциям на разделяющую прямую. Оказывается, что для каждой точки одной полуполосы нам нужно будет рассмотреть только те точки, которые попадают в прямоугольник размера $\delta \times 2\delta$. За счет того, что кратчайшее расстояние в множестве $S2$ не превышает δ, в прямоугольнике таких размеров может оказаться не более шести точек. Таким образом уточнить величину кратчайшего расстояния за один линейный проход по уже отсортированным спискам вершин каждой полуполосы (то есть за O(N)).

Чтобы достичь этой оценки на уточнение потребуется брать только параллельные разделяющие прямые, а сортировку провести заранее.

Таким образом, сложность $T(N)$ нахождения ближайшей пары точек, должна удовлетворять рекуррентной формуле:

$T(N) = 2\ T(N/2) + O(N)$.

Известно, что решением данного соотношения (а оно получилось таким же, как для сложности быстрой сортировки) является функция $T(N) = O(N \log N)$.

Задачи и разборы

Задача A. Выборы вождя

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Орки - одна из рас, населяющих мир Драэнор. Не отличаясь высоким интелектом, орки все же славятся своею силой и отвагой в бою. Ежегодно орки из разных кланов собираются в Долине Силы для того, чтобы избрать вождя всей Орды. В отличие от глупых людей, орки презирают выборы посредством голосования (да и, скажем прямо, все эти бюлетени, урны и избирательные участки чужды и непонятны орку, не державшему в руках ничего, кроме дубины и топора). Кандидаты в вожди сражаются друг с другом в честных поединках. В каждом поединке участвуют два претендента, один из которых выходит из него победителем, а другой оказывается поверженным. Проигравший в одном поединке орк выбывает из числа претендентов и не может участвовать в последующих поединках. Оставшийся в конце концов после всех боев кандидат и становит вождем Орды. Старейшины орков всегда наблюдают за выборами и любят предугадывать кто победит на них. Однако далеко не всегда можно предсказать не то, что общую победу на выборах, но даже победителя в одном конкретном бою. Конечно же все зависит от силы сражающихся - кто сильнее, тот и победит, однако в случае равенства сил может победить любой из них - тут уж как звезды станут.Старейшины обратились к вам с просьбой написать программу для определения количества претендентов, которые могут стать вождями.

Формат входного файла

В первой строке входного файла записано количество N претендентов на звание вождя в этом году ($1 \leq N \leq 1000000$), а во второй - N целых чисел в пределах от 1 до 10000, каждое из которых определяет силу соответствующего кандидата.

Формат выходного файла

Выходной файл должен содержать одно число - количество претендентов, которые могут стать вождями.

Пример

a.in	a.out
5 1 2 3 4 5	1
6 2 2 2 2 2 2	6
6 3 2 1 3 1 1	2

Разбор задачи A. Выборы вождя

Обозначим максимальную из сил всех орков – x. Нетрудно заметить, что стать вождем может только орк, который обладает этой наибольшей силой x. Действительно, пусть у нас есть орк с силой x. Тогда он может последовательно сразиться со всеми остальными орками. Тех, кто слабее его, он обязательно победит. А тех, кто имеет такую же силу, может победить при удачном для него исходе. Теперь рассмотрим орка с силой y, где $y<x$. В силу конечности множества орков, максимальную силу x имеет по крайней мере один из них. Для того, чтобы наш орк с силой y стал вождем либо нужно самому победить этого орка с максимальной силой, либо чтобы того кто-нибудь из претендентов “выбил” из соревнования. В силу неравенства $y<x$, первый вариант невозможен. Второй вариант возможен, но орка с силой x может выбить лишь орк опять же с максимальной силой, которого опять потребуется побеждать либо самому, либо за счет другого претендента. Но этот претендент должен быть тоже с максимальной силой x. Поскольку количество орков с максимальной силой конечно, то в конечном итоге останется только один орк с силой x, которого уже никто победить не сможет. А значит орк с силой y стать вождем никак не сможет.

Таким образом, достаточно найти в заданном массиве сил максимальный элемент, и затем определить сколько есть элементов, равных ему. При таком двухпроходном алгоритме потребуется либо сохранять вводимые из файла значения в массив, либо повторно прочитать файл. В случае, когда нет существенных ограничений на память, можно использовать первый вариант. Второй же вариант допустим, когда ввод-вывод файловый.

Если же используется ввод-вывод со стандартных устройств, и есть ограничения на память, не позволяющие сохранить целый массив, может быть использован следующий однопроходный алгоритм.

Будем хранить и обновлять две величины – максимальное значение в уже прочитанной части массива (max) и сколько раз оно там встречалось

(*count*). Начальные значения – когда прочитан первый элемент, *max* равен его значению, а *count*=1 (или еще до того, как что-то читать, *max*=∞, *count*=0). Когда читается очередной элемент, происходит следующее: если он меньше текущего максимума, ничего не меняем, если больше, то увеличиваем *max* до значения текущего элемента, а количество *count* делаем равным 1, в случае же равенства просто увеличиваем количество *count* максимальных элементов на 1.

Задача B. Покер

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

После нескольких походов и выигранных битв, у орков накопилось некоторое количество трофеев, которые вождь Оргрим Думхаммер должен распределить между воинами. Но хочет он сделать это в соответствии с тем, как они проявили себя в сражениях. Вождь приказал каждому орку принести 5 голов убитых им воинов противников и выложить их перед собой в ряд. Все бойцы выполнили этот приказ и теперь Оргрим должен оценить доблесть каждого. Возможны следующие оценки:

- нулевка (no pair) - все принесенные головы принадлежат воинам различных рас, то есть нет ни одной пары голов одной расы;
- пара (pair) - есть пара принесенных голов, принадлежащих воинам одной расы;
- две пары (two pair) - есть две пары одинаковых голов;
- тройка (set) - есть три головы воинов одной расы;
- фул-хаус (full house) - три головы одной расы, и две другой;
- каре (quads) - четыре головы одной расы;
- покер (poker) - все пять голов принадлежат воинам одной расы.

Оценки перечислены в порядке возрастания значимости и если комбинация голов у орка подходит под определение нескольких оценок, выбирается наиболее значимая. Помогите вождю написать программу, которая позволит ему оценить всех своих воинов по достоинству.

Формат входного файла

В первой строке входного файла содержится целое число T $(1 \leq T \leq 10000)$ - количество орков, принесших головы. В каждой из последующих T строк записаны через пробел 5 рас, которым принадлежат головы, выставленные соответствующим орком. Название каждой расы состоит не более чем из 7 маленьких латинских букв.

Формат выходного файла

В выходной файл нужно вывести T строк, в каждой из которых будет оценка комбинации голов соответствующего орка.

Примеры

b.in	b.out
4 elf elf human undead dwarf orc orc orc orc orc human goblin human goblin human gnome troll ogr morlock hobbit	pair poker full house no pair

Разбор задачи B. Покер

Аналогичная задача предлагалась на Московской школьной олимпиаде в 1986 году. Следующее довольно красивое (из-за своей простоты) решение взято из книги Брудно, Каплан “Московские олимпиады по программированию”.

Сосчитаем, сколько равных пар значений в данном массиве A (то есть количество выполнений равенства A[i]=A[j] для $1 \leq i < j \leq 5$). Число их (оказывается!) однозначно характеризует комбинацию:

количество равных пар	комбинация
0	нулевка
1	пара
2	две пары
3	тройка
4	фулхаус
6	каре
10	покер

Запрограммировать такой алгоритм не составляет труда.К сожалению, для большего пяти количества элементов подобное решение невозможно, поскольку уже для 6 карт есть комбинация “три пары”, которая имеет то же количество равных пар, что и “тройка” (а также “две тройки” дают такое же количество, как и “каре”).

В общем случае задача очевидным образом сводится к сортировке в исходном смысле этого слова (то есть разделение объектов на группы по виду или сорту, а в нашем случае просто по значению) и вычислению количества элементов в каждой группе. Но, как известно, эффективное решение задачи сортировки заключается в упорядочении исходного массива (расположении его элементов в порядке возрастания или убывания), в связи с чем именно процесс упорядочивания и называют сортировкой. Но для 5 карт такое решение кажется неоправданно громоздким.

Задача C. Каждый третий бесплатно

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Барлимен Баттербар - владелец небезызвестного трактира “Гарцующий пони”, расположенного в городке Бри. Именно сюда частенько наведываются уставшие после сражений орки, чтобы отведать кружечку-другую своего любимого напитка - гномоукладчика. Однако в последнее время стали появляться другие заведения, что привело к уменьшению количества клиентов трактира. Чтобы вернуть себе клиентов, Барлимен решил сделать в своем трактире акцию, что каждый заказавший определенное количество кружек гномоукладчика, получает еще один

бесплатно. Естественно, чем меньше будет это количество, тем более привлекательной будет эта акция для клиентов, но с другой стороны, если оно будет слишком маленьким, то хозяин не сможет получать прибыль (а может даже и будет терпеть убытки). Помогите трактирщику определить минимальное количество кружек, за которое может он давать бесплатную так, чтобы получать хоть какую-нибудь прибыль.

Формат входного файла

Входной файл может содержать данные нескольких тестов. Данные каждого теста находятся в одной строке и представляют собой два целых числа: цена закупки одной кружки гномоукладчика B и цена C, по которой трактирщик ее продает ($0 \le B < C \le 10^9$). Пара значений B=C=0 обозначает конец файла.

Формат выходного файла

В выходной файл необходимо ответ задачи для каждого теста в отдельной строке.

Примеры

c.in	c.out
5 8 100 1000 13 18 0 0	2 1 3

Разбор задачи C. Каждый третий бесплатно

Пусть x – количество кружек, за которые трактирщик дает одну бесплатную. За них он получит $x \cdot C$ монет, а их себестоимость составляет $(x+1) \cdot B$ монет. Чтобы иметь прибыль выручка должна быть строго больше затрат на закупку, то есть $x \cdot C > (x+1) \cdot B$.

Перенесем все члены с неизвестной x в левую часть – получим $x \cdot (C-B) > B$.

Поскольку $B<C$, то мы можем разделить обе части на $(B-A)$ и знак неравенства не изменится: $x > B/(C-B)$.

Таким образом нам необходимо найти наименьшее натуральное число, большее $B/(C-B)$.

Как известно, $\lfloor y \rfloor$ – это наибольшее целое число, не превосходящее y. Числа, меньшие $\lfloor y \rfloor$, очевидно будут и меньше y. Поэтому $\lfloor y \rfloor + 1$ обязательно является первым числом, которое превосходит y. Таким образом, ответ задачи равен B div $(C-B)$ + 1.

В операции div, согласно ограничения задачи, участвуют неотрицательные числа (и делитель ненулевой), поэтому никаких особенностей при записи этой формулы для машинной арифметики не возникает. Результат деления получается также неотрицательным, а значит после добавления единицы мы гарантированно получаем натуральное число.

Задача D. Распределение

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Для нападения на некоторые поселения людей, эльфов и карликов вождь Орды Оргрим Думхаммер сформировал из всех имеющих в наличии воинов N различных отрядов, которые были отправлены на завоевания. Однако прибывшие лишь только сейчас разведчики донесли о силах противников, скопленных в этих поселениях, что естественно скорректировало планы Оргрима. И теперь он хочет произвести перераспределение войск по отрядам, переводя воинов из одного отряда в другой. При этом, чтобы не создавать неразбериху в рядах своей армии и выполнить перераспределение как можно быстрее, количество таких переводов должно быть минимально возможным (за один раз переводится один солдат из некоторого отряда в другой). Напишите программу, которая определяет минимальное количество переводов для перераспределения войск.

Формат входного файла

Первая строка входного файла содержит целое число N $(1 \le N \le 10000)$ - количество отрядов. Вторая строка содержит изначальное распределение воинов по отрядам - N чисел, каждое из которых определяет количество воинов в соответствующем отряде. А в третьей строке - требуемое распределение солдат. Количество солдат в одном отряде не превышает 10^6. Гарантируется, что общее число воинов в изначальном распределении и требуемом совпадает.

Формат выходного файла

В выходной файл выведите минимально возможное количество переводов.

Примеры

d.in	d.out
3 5 8 10 5 8 10	0
2 6 7 5 8	1
5 1 2 3 4 5 2 3 4 5 1	4
5 1 2 3 2 1 0 0 9 0 0	6

Разбор задачи D. Распределение

Можно разделить отряды на те, в которых не хватает воинов, и на те, в который есть избыток (те отряды, в которых уже ровно столько воинов, сколько и нужно, мы трогать не будем). Ясно, что потребуется переводов не меньше, чем общая нехватка воинов в отрядах первого типа, поскольку за один перевод общая нехватка может уменьшаться не больше, чем на один. В силу условия баланса (общее число воинов в изначальном распределении и требуемом совпадает) суммарный избыток воинов в отрядах второго типа равен тому же количеству. И такого количества будет достаточно: пока у нас есть хотя бы один отряд с избытком орков, мы переводим одного воина в отряд с недостатком, уменьшая ровно на один как общую нехватку, так и избыток, и продолжаем до тех пор пока обе эти величины не станут равными 0. Таким образом, для решения задачи требуется посчитать суммарную разницу между изначальным количеством воинов и требуемым по всем отрядам, в которых первая величина больше второй. Единственным подводным камнем этой задачи является то, что при указанных ограничениях возможен результат, не вмещающийся в int (long), и даже unsigned, поэтому для результата следует использо-вать 64-разряный тип long long.

Задача E. Кладбище

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

После каждой великой битвы один из генералов орков Гром Хеллскрим считает своим долгом пойти на кладбище и воздать память погибшим в боях оркам. При этом он всегда обходит все могилы, начи-ная свой обход с могилы своего отца и заканчивая могилой матери. Все могилы выстроены в ряд и пронумерованы числами от 1 до N. Размер шага Грома позволяет ему перемещаться от одной могилы к другой, номер которой отличается не более чем на 2. При этом он не должен покидать пределов кладбища и не должен повторно посещать могилу, на которой уже побывал ранее при своем обходе. Напишите программу, помогающую Грому совершить обход всех могил с соблюдением указанных правил.

Формат входного файла

В единственной строке входного файла находятся три целых числа: общее количество могил на кладбище N, и номера могил отца и матери Грома A и B. ($2 \le N \le 10000, 1 \le A, B \le N, A \ne B$).

Формат выходного файла

В выходной файл необходимо ответ задачи. Ответ является последовательностью из $N - 1$ шагов, каждый из которых имеет формат <*знак*><*число*>, где <*знак*> - это один из знаков + или - и определяет направление очередного шага, а <*число*> равно 1 или 2 и определяет величину шага. В случае, если обход совершить невозможно, выведите число 0.

Примеры

e.in	e.out
5 2 4	-1 +2 +2 -1
4 2 3	0

Разбор задачи E. Кладбище

Данная задача (с измененной легендой) взята с XV командного чемпионата школьников Санкт-Петербурга по программированию. Автор задачи – Юрий Петров. Легенду исходного варианта задачи (“Болото 2”) составили Игорь Синев, Павел Маврин

Будем предполагать, что $A<B$. Если это не так, то можно их поменять местами и, получив путь, вывести его шаги в обратном порядке и с противоположными знаками (обратный путь). Либо сделать замену $A \to N - A + 1, B \to N - B + 1$ (симметрия относительно центра кладбища), и тогда только выводить шаги с противоположными знаками.

Итак, мы свели всю задачу к случаю $A<B$. План действий таков – обойдем все могилы левее A, так чтобы вернуться к A и, переступив через нее, затем дойдем до B, перешагнем ее и обойдем все, что находится правее B, в конечном итоге вернувшись к могиле B.

Обход всех могил левее A (а он необходим лишь в случае, когда $A>1$) можно осуществить единственным образом (так чтобы прийти к $A-1$ и перешагнуть на $A+1$. Сначала сделать шаг размера 2 влево, перешагнув могилу $A-1$, к которой нам надо вернуться. Продолжать идти с шагом 2 влево, пока это возможно, затем сдвинуться на одну могилу (влево или вправо) в зависимости от того возможен ли еще ход влево, и затем снова с шагом 2 идем вправо пока не достигнем $A-1$. От нее перешагиваем на $A+1$ и движемся до $B-1$ с шагом 1, и от этой последней могилы обходим все могилы справа от B аналогично тому, как обходили могилы левее A.

Решение невозможно, когда могилы A и B находятся на расстоянии 1 и при этом не являются крайними. В случае, когда A и B соседние, можно обойти либо часть левее A, либо часть правее B. А потом мы обязательно попадем в B. Поэтому, когда хотя бы одной из этих частей нет, обход возможен.

Еще одно решение, возможно чуть менее эффективное (за счет константы), но по видимому более простое в реализации заключается в том, чтобы каждый раз делать тот из допустимых ходов, который уводит нас как можно дальше от B.

Задача F. Продовольственная программа

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вернувшись из очередного похода, Оргрим Думхаммер обнаружил, что его поселение было полностью разрушено отрядами людей. Теперь ему придется заново отстраивать город, восстанавливая его из руин. Прежде всего, вождю Орды предстоит решить проблему с продовольствием - построить достаточное количество ферм (pig farm), чтобы его воины и рабочие не умерли от голода. Естественно, для этой работы Оргрим привлек всех своих пеонов (батраков). На постройку одной фермы требуется G_f единиц золота, при этом она обеспечивает продовольствием P_f орков. Однако каждая ферма должна строиться в непосредственной близости от какого-либо зала вождей (great hall). Поэтому необходимо будет создать предварительно некоторое количество залов вождей. В окрестности одного зала вождей может быть построено не более F ферм. При этом возведение каждого зала вождей требует G_h единиц золота, но зато может обеспечить продовольствием P_h орков.

Помогите Оргриму написать продовольственную программу, которая определит сколько нужно построить залов вождей и ферм для обеспечения продовольствием N орков, затратив на строительство минимальное количество единиц золота

Формат входного файла

Данные находятся в единственной строке входного файла и представляют собой шесть целых чисел N, F, G_h, P_h, G_f, P_f ($0 \le N \le 10^6, 1 \le F \le 100, 1 \le G_h, G_f \le 2000, 0 \le P_h, P_f \le 1000$, $P_f \ne 0$).

Формат выходного файла

В единственную строку выходной файл выведите три целых числа - количество залов вождей и ферм, которые необходимо построить, а также количество золота, которое необходимо затратить на это строительство.

Примеры

f.in	f.out
9 5 1200 1 500 4	1 2 2200
12 5 1200 1 500 4	1 3 2700
15 2 1000 0 550 5	2 3 3650

Разбор задачи F. Продовольственная программа

Пусть f и h - количество построенных ферм и залов вождей соответственно. Чтобы эти значения были решением задачи, они должны удовлетворять условиям:

- $f \geq 0, h \geq 0$ (количество построек не может быть отрицательным);
- $P_f \cdot f + P_h \cdot h \geq N$ (продовольствия должно хватать на N орков);
- $F \cdot h \geq f$ (количество ферм в окрестности одного зала вождей не должно превышать F);
- $G_f \cdot f + G_h \cdot h \to min$ (количество золота на постройки должно быть минимальным).

Все ограничения линейны, а переменные - целые, поэтому мы имеем дело с задачей целочисленного линейного программирования.

Можно было бы предположить, что нужное количество продовольствия нужно достигать за счет постройки новых ферм, а при отсутвии места для постройки очередной фермы возводить новый зал вождей. Такое решение могло бы быть реализовано даже за время $O(1)$. Если будет построено h залов вождей, то максимальное количество продовольствия, которое они могут обеспечить (вместе с F фермами вокруг каждого из них), будет равно $(P_f \cdot F + P_h) \cdot h$. Отсюда ясно, что наименьшее количество залов, которые обеспечат продовольствие N, будет равно $h = \lceil N/(P_f \cdot F + P_h) \rceil$ (округление вверх). Тогда остается выяснить сколько потребуется всего ферм, для чего воспользуемся неравенством $P_f \cdot f + P_h \cdot h \geq N$, из которого следует, что минимальное количество ферм, которые нужно построить при такой стратегии, будет равно $f = \lceil (N - Ph \cdot h)/P_f \rceil$. По видимому для всех известных стратегических игр этот алгоритм будет давать правильный результат. Но это так, лишь благодаря тому, что в них выполнены неравенства $P_f > P_h$ и $G_f < G_h$ (зал вождей (или его аналог) - структура более дорогая по сравнению с фермой, и при этом ферма дает больше продовольствия). В задаче же ничего подобного не гарантируется, и вполне

мыслима ситуация, когда залы вождей настолько дешевы, либо дают настолько много продовольствия, что выгоднее строить только их, а фермы не строить вовсе. Если бы значения f и h могли быть не только целыми, были бы возможны только два этих крайних случая: при строительстве отдавать преимущество той структуре, у которой ниже соотношение цена/качество (качество в данном случае обозначает объем продовольствия) – это следует также и из свойств задачи линейного программирования.

Для решения задачи целочисленного программирования в общем случае можно воспользоваться, например, методом Гомори. Его суть заключается в решении исходной задачи без требования целочисленности - тогда мы имеем дело с обычной задачей линейного программирования, для которой применим симплекс-метод. Однако оптимальный план может оказаться нецелочисленным, тогда согласно методу Гомори нужно “отрезать” его от множества допустимых планов дополнительным ограничением, сохраняющим однако все целочисленные планы задачи. В качестве такого ограничения может быть взято

$$\{a_{i1}\} \cdot x_1 + \{a_{i2}\} \cdot x_2 + \ldots + \{a_{in}\} \cdot x_n \geq \{b_i\},$$

где $\{\cdot\}$ - неотрицательная дробная часть, i - номер строки симплекс-таблицы, соответствующей переменной с наибольшей дробной частью в оптимальном плане, а b_i и a_{ij} - значения в i-ой строке симплекс-таблицы. После введения нового ограничения, мы снова решаем задачу симплекс-методом и продолжаем действовать таким образом до тех пор, пока не получится целочисленное решение.

Доказано, что метод Гомори сходится, то есть для нахождения оптимального целочисленного плана потребуется решение конечного количества задач линейного программирования. Однако какой-либо хорошей оценки этого количества в зависимости от размерности не получено (можно лишь получить оценку, зависящую от коэффициентов ограничений) - более того, есть даже двумерные задачи целочисленного программирования, для которых количество итераций в методе Гомори довольно велико. А поскольку с каждой итерацией увеличивается и размерность задачи (для очередного ограничения требуется введение дополнительной переменной, поскольку симплекс-метод работает лишь с ограничениями в виде равенств), то время работы всего алгоритма может оказаться неприемлемым. Еще одним (хотя и менее существенным) недостатком этого метода является необходимость работы с вещественными числами и связанное с этим накопление вычислительной погрешности (впрочем, это свойство характерно и для симплекс-метода).

Другой подход - использование метода ветвей и границ. Как и при решении задачи методом Гомори, первоначально находится симплексным методом оптимальный план задачи без учета целочисленности переменных. Если среди компонент этого плана нет дробных чисел, то тем самым найдено искомое решение данной задачи. Если же есть нецелые компонент, то необходимо осуществить упорядоченный переход к новым планам, пока не будет найдено решение задачи.

Предположим, что оптимальный план не удовлетворяет условию целочисленности переменных. Пусть, например, переменная x_i приняла дробное значение K_i. Тогда в оптимальном плане целочисленном плане ее значение будет либо меньше или равно $\lfloor K_i \rfloor$, либо больше или равно $\lfloor K_i \rfloor + 1$. Таким образом, исходная задача разбивается на две подзадачи: первая подзадача содержит по сравнению с исходной дополнительное ограничение $x_i \leq \lfloor K_i \rfloor$, а вторая - ограничение $x_i \geq \lfloor K_i \rfloor + 1$.

Найдем решение этих задач симплекс-методом. Возможен один из четырех случаев:

1. Одна из задач неразрешима, а другая имеет целочисленный оптимальный план. Тогда этот план - решение нашей исходной задачи.

2. Одна из задач неразрешима, а другая имеет оптимальный план, среди компонент которого есть дробные числа. Тогда дальше мы рассматриваем вторую задачу и снова разбиваем ее на две относительно какой-либо переменной, принявшей дробное значение на оптимальном плане.

3. Обе задачи разрешимы и по крайней мере одна из них имеет оптимальный целочисленный план. Если значение целевой функции на этом плане не превосходит оптимального значения для второй задачи (в случае когда обе задачи имеют оптимальными целочисленные планы, это условие очевидно выполнено), то найденный план является решением исходной задачи. Если же оптимальный план второй задачи - не целочисленный, и значение на нем меньше, чем у первой задачи (как минимум на 1), то следует опять же разбить ее на две подзадачи.

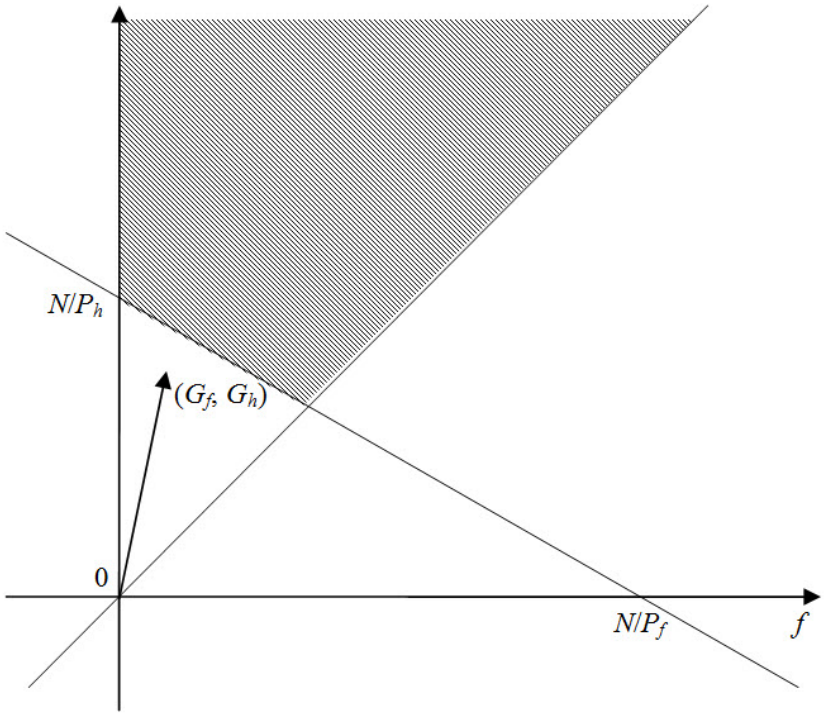
4. Последний (самый неприятный) случай - обе задачи разрешимы и их оптимальные планы не являются целочисленными. Тогда придется разбивать обе подзадачи. Но логичнее начать с той из них для которой значение целевой функции меньше. Именно ее мы и разобьем на две подзадачи относительно переменной с дробным значением.

Таким образом решение задачи может быть представлено как некоторый обход бинарного дерева, в котором корню соответствует исходная задача, а каждый узел является либо листом (если соответствующая задача имеет целочисленное решение, либо оптимальное значение целевое функции для нее не меньше, чем на уже найденном в какой-либо ветви целочисленном плане), либо имеет двух сыновей, соответствующих подзадачам на которые разбивается исходная. Если на некотором шаге будет получен целочисленный план, и значение на нем окажется не большим, чем значения целевой функции в других возможных для ветвления узлах, то данный план является оптимальным планом исходной задачи целочисленного программирования.

Этот метод по видимому имеет более предпочтительную оценку сложности в худшем случае по сравнению с методом Гомори. Это объясняется тем, что количество ограничений может увеличиться не больше чем на число переменных в исходной задаче. Кроме того, при таком подходе можно оставаться в рамках целочисленной арифметики. Конечно в методе ветвей и границ происходят ветвления, но можно показать что их количество ограниченно величиной зависящей лишь от размерности исходной задачи (правда, эта оценка будет экспоненциальной). Тем не менее, на практике (то есть в среднем) метод Гомори быстрее приводит к результату.

Рассмотрим теперь нашу конкретную двумерную задачу. Посмотрим, прежде всего, как будет выглядеть множество планов задачи. В общем случае в плоскости параметров (f, h) множество планов будет образовывать “трапецию”, большее основание которой неограниченно.

Поскольку вектор-градиент (G_f, G_h) целевой функции всегда направлен в первую четверть, то ее минимум (если не требовать целочисленности), достигается в точке $A(0, N/P_h)$ или в точке B пересечения прямых $P_f \cdot f + P_h \cdot h = N$ и $F \cdot h = f$. Рассмотрим участок прямой $P_f \cdot f + P_h \cdot h = N$, находящийся между точками A и B. Построим лучи, начинающиеся на этом участке в точке с целой координатой f и идущие вверх параллельно оси Oh. Все целочисленные точки этих лучей являются планами, но в силу того, что G_h положительно, оптимальной из них может быть лишь самая нижняя. Кроме того, еще одна точка является кандидатом на то, чтобы быть оптимальным планом - самая левая и нижняя точка над лучом (или на луче), ограничивающим множество планов, начинающимся в точке B.



При указанных в задаче ограничениях можно перебрать все эти точки и выбрать из них оптимальную. В терминах задачи мы начинаем с того, что у нас нет ни одной фермы и смотрим сколько нужно построить залов, чтобы обеспечить объем продовольствия N. Далее мы начинаем строить по одной ферме и опять же смотреть сколько потребуется залов. Для каждого варианта мы подсчитываем его стоимость. Все это имеет смысл делать лишь до тех пор, пока ферм не станет настолько много, что залы вождей нужно будет строить уже не для продовольствия, а для обеспечения возможности постройки ферм. Если бы мы продолжали дальше, увеличивалось бы как число ферм, так и число залов вождей (во всяком случае точно не уменьшалось бы), а значит росли бы и суммарные затраты на возведение этих структур.

Вычислительную сложность такого алгоритма можно оценить величиной порядка $O(N/P_f)$.

Тем не менее, если бы ограничения на N были бы выше, то указанный алгоритм мог бы не пройти по времени. Ключ к эффективному решению заключается в том, чтобы еще больше сократить количество рассматриваемых планов. Проведем через все целые точки, расположенные непосредственно над участком AB (или на нем самом), прямые параллельные прямой $P_f \cdot f + P_h \cdot h = N$.

Нетрудно увидеть, что общее количество таких прямых не превысит P_h. Действительно, параллельные прямые будут иметь уравнение -

$P_f \cdot f + P_h \cdot h = k$. Поскольку на этой прямой лежит по крайней мере одна целочисленная точка, то k обязательно целое. В силу того, что эта точка лежит не ниже прямой A, k должно быть не меньше N. Остается показать, что k не может быть больше $N + P_h$. Пусть некоторая точка (f, h), претендующая на оптимальность, лежит на прямой $P_f \cdot f + P_h \cdot h = k$, где $k \geq N + P_h$. Но тогда точка $(f, h - 1)$ лежит на прямой с уравнением, в правой части которого стоит $k - P_h$, что не меньше N, а это значит, что точка $(f, h - 1)$ является также целочисленным планом нашей задачи, а значение целевой функции на нем меньше, чем на (f, h). На каждой прямой $P_f \cdot f + P_h \cdot h = k$ мы можем перебирать не все целые точки, а проверить лишь крайние целые точки (причем то, на каком конце (левом или правом) прямой будет достигаться оптимальное значение, по прежнему легко определяется соотношением цена/качество).

Для нахождения целочисленной точки на каждой прямой нам потребуется единожды найти решение в целых числах уравнения $P_f \cdot f + P_h \cdot h$ = НОД(P_f, P_h), что займет по времени $O(log\ min(P_f, P_h))$, если использовать алгоритм Евклида, и затем для $P_h/$ НОД(P_f, P_h) значений переменной k нужно будет находить крайнюю целую точку прямой $P_f \cdot f + P_h \cdot h = k$. Для каждого конкретного k при известном уже решении уравнения с НОД, это может быть выполнено за $O(1)$.

Не будем выписывать полностью формулы для этого подхода, а используем другую идею. По прежнему будем перебирать возможные значения f, но теперь будем делать это с того конца, где достигается минимум не целочисленной задачи. Количество требуемых залов вождей будет каждый раз определяться по формуле $h = \lceil (N - P_f \cdot f)/P_h \rceil$. В случае, когда остаток от этого деления станет таким же как в начальной точке, мы попадем на ту же самую прямую, на которой находились вначале, и потому дальше уже новых прямых не возникнет, и все крайние точки будут перебраны. Сложность алгоритма - $O(N/P_f, P_h/$ НОД$(P_f, P_h))$.

Отдельного внимания требует случай, когда $P_h = 0$. Тогда прямая $P_f \cdot f + P_h \cdot h = N$ проходит параллельно оси Oh и “трапеция” вырождается в “треугольник”. Но этот случай довольно просто рассматривается даже в терминах задачи. Поскольку продовольствие обеспечивают нам только фермы, то их необходимо построить в количестве $f = \lceil N/P_f \rceil$ штук, а залов вождей потребуется $h = \lceil f/F \rceil$.

Задача G. Призыв в армию

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Для похода на Азерот Оргриму Думхаммеру потребовался еще один отряд. На призыв явились N орков. Способности в ближнем бою и метании копья каждого из них Оргрим сразу же оценил. Теперь же он должен определить кого из них назначить солдатом-пехотинцем (grunt), а кого метателем-охотником за головами (headhunter). При этом, для того, чтобы отряд был боеспособным, необходимо, чтобы было в отряде было не менее G грунтов и не менее H хедхантеров. После определения каждого орка в какой-то тип войск, может быть определена сила этого отряда, как сумма способностей всех орков в назначенной им специализации. Напишите программу, определяющую максимально возможную силу вновь призванного отряда

Формат входного файла

В первой строке входного файла заданы три целых числа $N, G, H (1 \leq N \leq 10000, 1 \leq G, H \leq N)$. Далее идут N строк, в каждой из которых за-писаны два целых числа в диапазоне от 0 до 10000 - способность соответствующего орка в ближнем бою и его способность в метании копья.

Формат выходного файла

В выходной файл необходимо максимальную силу боеспособной армии, которая может быть создана из призывников. В случае невозможности создания армии, удовлетворяющей условиям, выведите число -1.

Примеры

g.in	g.out
4 0 0 1 2 2 1 3 4 4 3	12
3 1 1 5 5 6 7 6 8	20
2 2 2 4 5 2 3	-1
3 0 3 5 2 6 1 7 0	3

Разбор задачи G. Призыв в армию

Очевидно, что решение задачи невозможно только в том случае, когда $G+H > N$. В этом случае нет ни одного варианта создать отряд, поскольку обязательно требуется больше орков, чем имеется в наличии. Когда $G+H$ не превосходит N, то существует вариант выбора назначения каких-нибудь G орков грунтами, а H - хедхантерами, оставшихся можно назначать как угодно. Поэтому решение существует, следует только выбрать из всех таких вариантов оптимальный. Пусть g_i - сила i-го орка в ближнем бою, а h_i - сила i-го орка в метании копья.

Докажем, что оптимальное назначение следует искать среди тех, которые содержат G орков с максимальными (по сравнению с остальными) значениями $g - h$ в качестве грунтов. Предположим, что найдено некоторое допустимое назначение, в котором среди из G орков с максимальным значением величины $g - h$ есть по крайней мере один орк назначенный метателем (пусть его номер j). Тогда среди остальных $N - G$ орков должен быть по крайней мере один, назначенный пехотинцем (пусть его номер k). В силу выбора j и k выполяется неравенство $g_j - hj \geq g_k - h_k$. Сделаем переназначение: орка с номером j сделаем грунтом, а с номером k - хедхантером. Общее количество грунтов и хедхантеров в отряде не изменится, а сила отряда увеличится на величину $g_j + h_k - h_j - g_k \geq 0$. Это

значит, что любое назначение не ухудшится, если мы выполним несколько переназначений так, чтобы все G орков с максимальным значением $g - h$ стали грунтами.

Аналогичным образом доказывается, что H орков с минимальными значениями $g - h$ должны быть взяты в качестве хедхантеров. Остальные орки могут быть назначены как хедхантерами, так и грунтами - естественно для увеличения силы нужно определять их в тот тип войск, в котором они будут сильнее. Для эффективного решения этой задачи потребуется упорядочение всех орков по величине $g - h$.

Задача H. Родственники

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Кроме того, что имена орков невероятно выразительны, они еще могут сказать кое-что о родственных связях их обладателей. Например, если два орка носят имена, которые могут быть сделаны одинаковыми посредством удаления не более половины букв из каждого, то эти орки являются родственниками в первом колене. Так, например, orgrim и grom являются родственниками в первом колене, поскольку из первого имени могут быть удалены три буквы из шести (o,r,i), а из второго – одна из четырех (o), и тогда получатся одинаковые имена grm. Двое орков с именами *A* и *B* будут родственниками в (*n+1*)-ом колене, если существует такое имя *C*, что орки с именами *A* и *C* являются родственниками в 1 колене, а *B* и *C* – родственниками в *n*-ом колене. Нетрудно заметить, что если два орка являются родственниками в *n*-ом колене, то они являются родственниками и в любом колене $m > n$. Однако степень родства орков определяется наименьшим коленом, в котором они являются родственниками.

Напишите программу, которая для двух орков с заданными именами определяет степень $n\,(n \geq 1)$ их родства.

Формат входного файла

Входной файл описывается двумя строками, состоящими из маленьких латинских букв и определяющих имена двух орков. Каждое имя состоит не менее чем из 1 буквы и не более чем из 100.

Формат выходного файла

В выходной файл выведите степень n родства орков с заданными именами или строку “not related”, если данные два орка не являются родственниками в n-ом колене ни при каком n.

Примеры

h.in	h.out
orgrim grom	1
a b	2
goo ogg	2

Разбор задачи H. Родственники

Данная задача (с незначительными изменениями в легенде) взята с Waterloo local contest (24.09.2006, Problem E). Автор задачи – Gordon V. Cormack, исходное название – Cousins.

Сразу заметим, что при указанных в задаче ограничениях ответ “not related” невозможен. Ясно, что любые два орка с однобуквенными имени являются родственниками не более, чем во втором колене (их общим родственником в первом колене является орк, имя которого состоит из двух букв, первая из которых принадлежит первому орку, а вторая – второму (либо наоборот)). Кроме того, от любого орка с непустым именем можно “протянуть” цепочку родственников к имени, состоящему из одной (скажем, первой) буквы (последовательно удаляя по крайней мере по одной букве). Таким образом, каждый орк является родственник орку, имя которого – его первая буква. А все однобуквенные орки, как мы уже определили, родственники. Значит любые два орка с непустыми именами s_1 и s_2 находятся между собой в родственной связи. Для колена же этой связи можно получить очень грубую, но все равно довольно оптимистичную оценку сверху – $l_1 + l_2$, где $l_1 = \text{length}(s_1)$ и $l_2 = \text{length}(s_2)$ – длины имен орков.

Если бы были допустимы пустые имена, то был бы возможен ответ “not related”, поскольку орк с пустым именем может быть родственником только орку так же с пустым именем.

Отдельно стоит упомянуть о том, что орки с одинаковыми именами являются родственниками в первом колене, поскольку по условию n всегда

не меньше 1.

Перейдем к решению. Пусть $l_1 \geq l_2$ (если это не так их можно поменять местами). Из определения следует, что два орка являются родственниками в первом колене, если их имена содержат общую подпоследовательность длины не меньше, чем половина длины большего имени. Если ввести обозначение LCS(s_1,s_2) – наибольшая общая подпоследовательность (longest common subsequence) строк s_1 и s_2, то предыдущее условие может быть записано как length(LCS(s_1, s_2)) $\geq \lceil l_1/2 \rceil$.

Рассмотрим теперь условия, при которых s_1 и s_2 будут родственниками во втором колене. Для этого должна существовать такая строка s (пусть ее длина равна l), что length(LCS(s_1, s))$\geq \lceil \max(l_1, l)/2 \rceil$ и length(LCS(s_2, s))$\geq \lceil \max(l_2, l)/2 \rceil$. Очевидно, что LCS(LCS($s_1, s$), LCS($s_2, s$)) – это наибольшая общая подпоследовательность строк s, s_1 и s_2, и поэтому она является подпоследовательностью LCS(s_1, s_2) (строго говоря, это рассуждение некорректно, поскольку LCS определяется не вполне однозначно; более корректно говорить о том, что для любой максимальной подпоследовательность s_1 и s и существует такая максимальная подпоследовательность s_2 и s, что их наибольшая подпоследовательность содержится в некоторой LCS(s_1,s_2), но наверное это лишь затруднит понимание).

Допустим LCS(LCS(s_1,s),LCS(s_2,s)) не равно LCS(s_1, s_2). Это означает, что LCS(s_1, s_2) не содержится в качестве подпоследовательности в s. Но тогда за счет добавления в s не содержавшейся в ней ранее (в соответствующей позиции) буквы из LCS(s_1, s_2) мы сможем удлинить на этот символ как LCS(s_1, s), так и LCS(s_2, s). Правые части неравенств length(LCS(s_1, s)) $\geq \lceil \max(l_1, l)/2 \rceil$ и length(LCS(s_2,s)) $\geq \lceil \max(l_2, l)/2 \rceil$ увеличатся на 1, а левые либо не изменятся, либо увеличатся на 1.

Отсюда следует, что в качестве общего родственника s имеет смысл брать только те строки, которые содержат LCS(s_1,s_2).

Пусть length(LCS(s_1,s_2))=k. Возьмем сначала s как LCS(s_1,s_2). Мы можем взять некоторое количество (допустим k_1) символов из строки s_1 и вставить их в s в такие позиции, чтобы s оказалась подпоследовательностью s_1, а это значит, что длина LCS(s_1,s) будет равна k+k_1. LCS(s_2,s) при этом останется по прежнему равным k (в противном случае, поскольку LCS(s_2,s) является также и подпоследовательностью s_1, не было бы верным равенство length(LCS(s_1,s_2))=k). Добавим таким же образом k_2 символов из строки s_2 к строке s, чтобы LCS(s_2,s) стало равным k+k_2. Другие символы добавлять не имеет смысла, так как они не изменят LCS(s_1,s) и LCS(s_2,s), но увеличат длину строки s, что может привести лишь к увеличению величин $\lceil \max(l_1, l)/2 \rceil$ и $\lceil \max(l_2, l)/2 \rceil$.

Остается только перебрать все возможные варианты для k_1 и k_2 (в этом месте можно заметить, что конкретные символы в строках не важны, важны лишь длины самих строк и длины общих подстрок).

Возможные варианты для этих величин – ($0 \le k_1 \le l_1 - k$, $0 \le k_2 \le l_2 - k$). При этом l получается равным $k + k_1 + k_2$. length(LCS(s_1, s))=$k + k_1$, length(LCS(s_1, s))=$k + k_2$.

Учитывая это, перепишем неравенства, гарантирующие, что s_1 и s_2 являются родственниками во втором колене в виде:

$$k + k_1 \ge \lceil l_1/2 \rceil ,$$

$$k + k_1 \ge \lceil (k + k_1 + k_2)/2 \rceil ,$$

$$k + k_2 \ge \lceil l_2/2 \rceil ,$$

$$k + k_2 \ge \lceil (k + k_1 + k_2)/2 \rceil .$$

Из первого и третьего неравенства получаем, что k_1 должно быть не меньше $\lceil l_1/2 \rceil - k$, k_2 – не меньше $\lceil l_2/2 \rceil - k$, а из второго и четвертого, что разность между k_1 и k_2 не должна превышать по модулю k.

Таким образом, необходимо проверить, чтобы на отрезках $\lceil l_1/2 \rceil - k \le k_1 \le l_1 - k, \lceil l_2/2 \rceil - k \le k_2 \le l_2 - k$, отыскались значения, отличающиеся друг от друга не более, чем на k. Допустим нашлась такая пара значений k_1 и k_2, удовлетворяющая указанным ограничениям. Если k_1 больше, чем $\lceil l_1/2 \rceil - k$, то можно уменьшить его значение на 1. Если исходное k_2 было меньше чем k_1+k, то ничего менять не нужно – новая разность между k_2 и k_1 не превысит k. Если же исходное значение k_2 было равно k_1+k, то его следует также уменьшить на 1 (благодаря условию $l_1 \ge l_2$, это возможно сделать, не нарушив ограничения $\lceil l_2/2 \rceil - k \le k_2 \le l_2 - k$). Таким образом всегда можно добиться того, чтобы k_1 было равно левой границе $\lceil l_1/2 \rceil - k$. А это значит, что достаточно проверить, чтобы было выполнено неравенство $(\lceil l_1/2 \rceil - k) - (l_2 - k) \le k$. Или после преобразования – $\lceil l_1/2 \rceil \le l_2 + k$.

На языке строк это можно выразить следующим образом – мы берем в качестве s строку LCS(s_1,s_2) дополняем ее $\lceil l_1/2 \rceil - k$ символами строки s_1, так что LCS(s_1, s) станет равным $l_1/2$. Теперь мы можем добавить к s не более $l_1/2$ символов из строки s_2 (чтобы из s можно было удалить не больше половины символов для получения LCS(s_1,s)), с другой стороны их всего в s_2 есть l_2-k. Поэтому из строки s_2 в s добавим $\min(\lceil l_1/2 \rceil , l_2 - k)$ символов. Теперь возникает вопрос: можно ли из строк s и s_2 получить удалением не более половины символов строку LCS(s_2,s)? В случае, когда минимум в выражении $\min(\lceil l_1/2 \rceil , l_2 - k)$ достигается на $\lceil l_1/2 \rceil$, LCS(s_2,s) будет иметь длину $\lceil l_1/2 \rceil + k$. В строке s будет $2\lceil l_1/2 \rceil$ символов, и для

получения LCS(s_2,s) нужно будет удалить $\lceil l_1/2 \rceil - k$, что очевидно не больше половины длины s. В строке же s_2, имеющей длину l_2, нужно оставить $\lceil l_1/2 \rceil + k$ символов. Но, благодаря неравенству $l_1 \geq l_2$, это количество будет точно не меньше половины l_2. Если же минимум достигается на $l_2 - k$, то в строке LCS(s_2,s) будет l_2 символов (то есть LCS(s_2,s)=s_2). Тогда для получения LCS(s_2,s) из строки s_2 вообще ничего удалять не нужно, а из строки s потребуется удалить $\lceil l_1/2 \rceil - k$ символов, и это значение не должно превышать l_2 (то количество символов, которые мы хотим оставить). Отсюда сразу же получается неравенство $\lceil l_1/2 \rceil \leq l_2 + k$.

Фактически здесь мы находим из всех родственников s_1 в первом колене такого s, который наиболее "близок" к s_2 (для него длина LCS(s_2,s) – максимальная из возможных). Если он окажется непосредственным родственником s_2, то s_1 и s_2 будут родственниками во втором колене. Если же нет, то потребуется определить степень родства s_2 и s. А степень родства s_1 и s_2 будет тогда на 1 больше. Отсюда получается достаточно простой алгоритм для определения степени родства.

Чуть более сложный в вычислительном плане, но по видимому более простой для понимания алгоритм получится, если мы начнем от одного из данных нам орков, найдем всех его родственников, затем для каждого из них – всех родственников во втором колене, и так далее, пока не дойдем до того, что будет найдено имя второго орка. То есть обычный поиск в ширину. Но учитывая, что возможных строк невероятно много, прямая реализация этого алгоритма обречена в лучшем случае на "memory-limit", а скорее всего и "time-limit". Тем не менее, мы можем вспомнить, что для определения родства в первом колене важна лишь информация о длине имени и длине LCS данной строки с искомой. В таком случае поиск в ширину мы можем вести уже в пространстве пар (length(s), length(LCS(s,s_2))), отождествляя тем самым строки для которых эти величины совпадают. Но это множество пока что тоже не ограничено: если length(LCS(s,s_2)) очевидно не превосходит длины строки s_2, то на length(s) ограничений нет. Однако, если учесть то, что нет смысла рассматривать строки не содержащие LCS(s_1,s_2), а также к ней добавлять символы которые не входят ни в строку s_1, ни в строку s_2, то мы сможем получить такое ограничение: $k \leq \text{length}(s) \leq k + (l_1 - k) + (l_2 - k)$. Теперь уже можно запускать поиск в ширину для нахождения длины кратчайшего пути по родственникам от пары (k,l_1) до пары (l_2,l_2). При попадании в очередной узел мы можем перебирать все возможные варианты для k_1 и k_2 (количество символов которые добавляются к текущему состоянию из строк s_1 и s_2 соответственно).

Оценка сложности такого алгоритма будет O(L^4), где L – длина наибольшей строки. При указанных ограничениях этого достаточно чтобы уло-

житься в отведенное время.

Остается напомнить классический алгоритм нахождения длины LCS. Он основан на принципе динамического программирования. Определим функцию, которую можно рекуррентно вычислять и которая позволить нам решить нашу задачу:

1. *Функция*. Пусть $f(i_1,i_2)$ – длина LCS префиксов строк s_1 и s_2 длин i_1 и i_2 соответственно (то есть $s_1[1{:}i_1]$ и $s_2[1{:}i_2]$). Область определения – $0 \le i_1 \le l_1, 0 \le i_2 \le l_2$. (То есть, префиксы могут быть и пустыми).

2. *Выражение для ответа на задачу через функцию*. То, что требуется найти, то есть длина LCS(s_1,s_2), – это просто $f(l_1,l_2)$.

3. *Рекуррентная формула*. $f(i_1,i_2)=f(i_1\text{-}1,i_2\text{-}1)+1$, если $s_1[i_1]=s_2[i_2]$, и $f(i_1,i_2)=\max(f(i_1,i_2\text{-}1),f(i_1\text{-}1,i_2))$, если $s_1[i_1]\neq s_2[i_2]$.

4. *Начальные значения*. $f(i_1,i_2)=0$, когда i_1 или i_2 равно 0 (то есть когда хотя бы одна из строк – пустая).

Для написания программы необходимо завести массив, в котором будут храниться значения нашей функции. Записать в соответствующие элементы начальные значения, а затем все остальные элементы заполнить по рекуррентной формуле.

Задача I. Деление земель

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Война орков с силами Альянса завершилась абсолютной победой Орды. Лордерон лежит в руинах - кровожадные полчища орков под предводительством Оргрима Думхаммера прошли по провинциям Лордерона и гномьему королевству Хаз-Модан, сметая все на своём пути. В землях Азерота теперь наступило господство Орды. Пришло наконец время разделить завоеванные земли между кланами Орды. И вот генералы, представляющие разные кланы, склонились над картой, проводя прямые линии, которые будут границами нового административного деления.

Напишите программу для определения количества земель, на которые эти линии разобьют карту Лордерона.

Формат входного файла

В первой строке входного файла записаны размеры карты W и H (левый нижний угол имеет координаты $(0, 0)$, правый верхний - (W, H)) и количество проведенных прямых N ($1 \leq W, H \leq 100, 1 \leq N \leq 1000$). В каждой из последующих N строк заданы коэффициенты a, b, c в уравнении соответствующей прямой $ax + by + c = 0$ ($-1000 \leq a, b \leq 1000, -10^6 \leq c \leq 10^6, |a| + |b| \neq 0$). Все числа во входном файлы - целые.

Формат выходного файла

В выходной файл необходимо вывести количество земель, образовавшихся на карте.

Примеры

i.in	i.out
8 6 4 0 2 -3 0 -1 4 1 0 -3 2 1 -10	9
10 10 1 1 2 3	1
3 5 2 0 1 -2 0 -2 4	2

Разбор задачи I. Деление земель

Для решения этой задачи можно воспользоваться формулой Эйлера для планарных подразбиений плоскости (очевидно, что разбиение ограниченной части плоскости конечным множеством прямых является планарным подразбиением). Если В - количество его вершин, Р - количество ребер, а Г - число граней, то всегда выполняется В - Р + Г = 1. Нетрудно определить количество вершин - для этого необходимо определить точки пересечения для всех пар прямых (с учетом также прямых определяющих края карты). Удалив повторы и точки находящиеся за пределами карты, мы получим все вершины. Для получения же количества ребер, необходимо посмотреть сколько из вершин принадлежат каждой прямой. Если

на некоторой прямой s вершин, то на ней находится $s-1$ ребро, а ребра, принадлежащие различным прямым, естественно не могут совпадать. Имеет смысл хранить списки вершин, принадлежащих каждой прямой. Тогда последовательно обрабатывая прямые, мы будем добавлять вершины, в которых текущая прямая пересекает какую-либо другую, в списки для обоих этих прямых, если конечно в этих списках эта вершина не встречалась уже. Реализация этих списков в виде массивов или именно списков потребует порядка $O(N)$ операций на точку для проверки того, есть ли уже такая точка, и $O(1)$ для добавления. Поскольку общее количество точек может достигать величины порядка $O(N^2)$, алгоритм будет иметь временную сложность $O(N^3)$, что при N равном 1000 может оказаться достаточно большой величиной. Если же реализовать списки точек на одной прямой в виде множеств (красно-черных деревьев), то оба действия (добавление и поиск) будут занимать по времени $O(\log N)$. Тем самым мы получим оценку времени работы всего алгоритма - $O(N^2 \log N)$. Однако здесь есть некоторые технические моменты, которые стоит оговорить. Несмотря на то, что прямые имеют целочисленные коэффициенты (a, b, c), координаты точек их пересечения могут быть не целыми. В связи с этим возникает вопрос как определять совпадают ли точки - может оказаться так, что координаты в действительности одинаковых точек будут отличаться на некоторую малую величину за счет вычислительной погрешности. С другой стороны при сравнении с некоторой точностью может оказаться так, что точки довольно близкие друг к другу будут засчитаны как одинаковые. Это может случиться также и из-за точности, обеспечиваемой выбранным вещественным типом. Хотя координаты точек и не целые, но они будут рациональными, причем знаменатели у обоих дробей, представляющих координаты будут одинаковыми. Удобно воспользоваться проективной плоскостью, на которой каждая точка (как и прямая) задается тремя координатами, причем точка (a, b, c), у которой с не равно 0, соответствует точке $(a/c, b/c)$ на обычной плоскости. При этом мы отождествляем все точки лежащие на прямой, проходящей через начало координат. Для такой плоскости легко получить координаты точки пересечения двух прямых - достаточно найти векторное произведение векторов их коэффициентов. Аналогично, для нахождения прямой проходящей через две заданные точки нужно векторно перемножить эти точки. Чтобы точки можно было легко сравнивать на совпадение простым сравнением всех компонент, нужно выбрать только одного представителя из всех “целочисленных” точек проективной плоскости. Для этого разделим все компоненты на их НОД, и сделаем (за счет смены знака всех координат) первую из ненулевых компонент положительной. Однако для множества в STL требуется не только возможность проверки на равенство, но еще и отношение “меньше”. Поскольку нам не важен

точный порядок, в котором должны следовать вершины, а только лишь размер множества, то вполне можно выбрать лексикографический порядок для векторов координат выбранных представителей точек на проективной плоскости. Есть еще один подход к решению этой задачи - имеющий ту же оценку по времени, но более экономный по памяти. Эта идея может быть использована, в частности, и при доказательстве формулы Эйлера. Пусть у нас уже есть некоторое разбиение первыми k прямыми, и уже вычислено количество граней в этом разбиении. Тогда возьмем очередную прямую и найдем количество s различных точек пересечения этой прямой с ранее рассмотренными k прямыми, попадающих на карту. Количество ребер, которые принадлежат этой прямой, будет равно $s-1$, и каждое из этих ребер проходит по одной из существовавших ранее граней, разбивая ее на две. И значит, количество граней увеличится на $s-1$. При таком подходе достаточно хранить лишь множество вершин, принадлежащих текущей прямой, что позволит снизить сложность по памяти с $O(N^2)$ до $O(N)$. В задаче есть еще один подводный камень, высвеченный однако одним из примеров входных данных - возможное существование нескольких уравнений задающих одну и ту же прямую. Ясно, что если некоторая прямая встречалась ранее, то второй раз ее рассматривать не нужно.

Задача J. Экстремальные расстояния

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

На плоскости задано множество, состоящее из N точек. Напишите программу, определяющую наибольшее и наименьшее из всех попарных расстояний между различными точками.

Формат входного файла

В первой строке входного файла записано количество точек N ($2 \le N \le 100000$). В каждой из последующих N строк содержится по два целых числа, по модулю не превышающих 10^6, которые определяют координаты соответствующих точек.

Формат выходного файла

Выходной файл должен содержать два числа с точностью не менее двух знаков после запятой - максимальное и минимальное расстояние соответственно.

Примеры

j.in	j.out
5 -2 0 0 0 3 0 1 0 -5 0	8.00 1.00
4 0 0 1 0 1 1 0 1	1.41 1.00

Разбор задачи J. Экстремальные расстояния

Разобрана в лекции.

Задача K. Наименьший квадрат

Имя входного файла:	k.in
Имя выходного файла:	k.out
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Во дворце великого султана Сулеймана Великолепного произошло ужасное происшествие - на полу образовалось черное пятно, имеющее форму прямоугольника. Придворные поговаривают, что это проклятие магрибского колдуна, которого великий султан не пожелал принять должным образом. Везирь посоветовал закрыть это пятно золотой плитой. Зная как султан почитает точность и красоту, везирь потребовал, чтобы эта плита имела форму квадрата. Но как ни велика государственная казна, но лишних динаров в ней нет. А это значит, что нужно потратить как можно меньше золота для изготовления плит. Напишите программу, которая находит квадрат наименьшей площади, который полностью покрывает заданное прямоугольное пятно

Формат входного файла

Входной файл состоит из 4 строк, в каждой из которых содержится по два вещественных числа - координаты соответствующей вершины прямоугольника. Вершины задаются в порядке обхода по контуру. Все числа

заданы с точностью до двух знаков после запятой и не превосходят по модулю 1000.

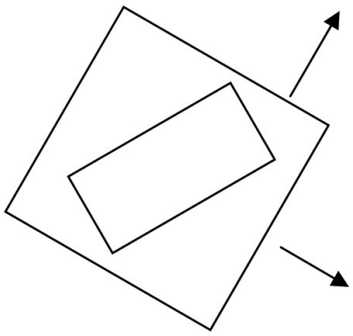
Формат выходного файла

В выходной файл выведите (в том же формате) координаты квадрата, который содержит данный прямоугольник. Вывод должен осуществляться с точностью не менее трех знаков. Если может быть построен квадрат с координатами, отличающимися от выведенных не более, чем на 0.0005, и его площадь будет минимальной, то ответ будет считаться верным.

Примеры

k.in	k.out
0.00 0.00 3.00 0.00 3.00 2.00 0.00 2.00	0.000 2.600 0.000 -0.400 3.000 -0.400 3.000 2.600
1.30 2.00 1.60 1.10 3.10 1.60 2.80 2.50	1.200 2.300 1.700 0.800 3.200 1.300 2.70 2.80

Разбор задачи K. Наименьший квадрат

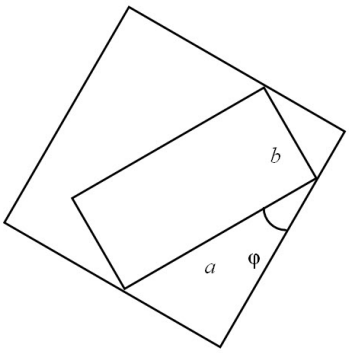


Очевидное на первый взгляд решение – продлить меньшую сторону прямоугольника до размеров большей, получив таким образом квадрат – оказывается неверным. Например, в частном случае, когда прямоугольник вырождается в отрезок (меньшая сторона близка к нулю, выгоднее использовать квадрат, для которого этот отрезок является диагональю.

Прежде всего докажем, что по крайней мере три вершины прямоугольника должны находиться на сторонах оптимального квадрата. Возьмем минимальный квадрат. Если ни одна из вершин прямоугольника не принадлежит его границе, то сначала параллельным переносом вдоль направления, задающегося одной из сторон прямоугольника, мы получим касание одной из вершин прямоугольника стороны квадрата. Затем осуществим параллельный перенос вдоль направления другой стороны. Таким образом, не изменив площади квадрат мы добьемся того, что две соседние вершины прямоугольника будут лежать на двух смежных сторонах квадрата. Если ни одна из оставшихся вершин не лежит на границе

квадрата, то можно уменьшать (это показывает, что выбранный квадрат не мог быть оптимальным) на одну и ту же величину обе стороны квадрата до тех пор, пока еще хотя бы одна вершина прямоугольник не коснется стороны квадрата. Тогда мы получим три вершины на границе квадрата, который не хуже исходного. Это завершает доказательство.

Пусть одна из сторон квадрата содержит вершину прямоугольника и образует с большей его стороной угол φ. Обозначим большую сторону прямоугольника a, а меньшую – b.



Тогда сторона квадрата должна быть не меньше каждой из величин $a\cos\varphi+b\sin\varphi$ и $a\sin\varphi+b\cos\varphi$. Нетрудно проверить, что когда угол φ не превышает 45 градусов, то больше первая величина, а при больших – вторая. Кроме того, в случае когда угол φ больше 45 градусов, замена переменных $\varphi \to 90-\varphi$ приводит нас к рассмотрению той же величины $a\cos\varphi+b\sin\varphi$ на отрезке $\varphi \in [0;45]$.

Возьмем вторую производную:

$$\frac{d^2}{d\varphi^2}(a\cos\varphi+b\sin\varphi) = -a\cos\varphi-b\sin\varphi.$$

Поскольку вторая производная на рассматриваемом отрезке отрицательна, то функция выпукла и потому не может иметь внутреннего минимума. И, значит, минимум выражения достигается на одном из концов.

При $\varphi = 0$ получаем просто a, а при $\varphi = 45$ – $(a+b)/\sqrt{2}$. Остается лишь проверить, где достигается минимум, и построить соответствующий квадрат.

Задача L. Вражда

Имя входного файла:	`l.in`
Имя выходного файла:	`l.out`
Ограничение по времени:	4 с
Ограничение по памяти:	256 Мб

В связи с очередным повышением цены на российский газ, Украина увеличила плату России за транзит российской нефти в Европу. В ответ на это Государственная Дума Российской Федерации решила построить новый нефтепровод “Вражда”, который пройдет в обход украинской территории. Вам предстоит спроектировать начальный участок этого нефтепровода. Имеется N различных месторождений нефти, которые задаются

своими координатами на плоскости. Необходимо провести трубу по прямой таким образом, чтобы сумма длин дополнительных труб, идущих от месторождений до основной трубы, была минимальной. Диаметром трубы можно пренебречь.

Формат входного файла

Входной файл содержит количество месторождений N ($1 \leq N \leq 2009$). Далее следуют N строк, в каждой из которых записана пара целых чисел x, y - координаты соответствующего месторождения ($-10000 \leq x, y \leq 10000$).

Формат выходного файла

В выходной файл необходимо вывести минимально возможную суммарную длину всех дополнительных труб, идущих от месторождений до основной трубы с точностью не менее двух знаков после запятой.

Примеры

1.in	1.out
4 -1 -1 1 1 -1 1 1 -1	2.83

Разбор задачи L. Вражда

Докажем, что искомая прямая должна проходить через 2 точки данного множества. Пусть у нас есть прямая a, которая не проходит ни через одну точку множества. Тогда некоторое количество точек (допустим k) лежит в положительной полуплоскости относительно этой прямой, а остальные ($N - k$) – в отрицательной. Перемещая прямую на небольшое расстояние x в направлении нормального вектора к прямой, мы уменьшаем расстояния до каждой из точек положительной полуплоскости на величину x, и увеличиваем расстояния до каждой точки отрицательной полуплоскости на ту же величину. Если $k > N-k$, то выполнив указанный сдвиг мы уменьшим суммарное расстояние от всех точек до нашей прямой. Если $k < N - k$, то следует перемещать прямую в обратном направлении. Если же $k = N - k$, то сдвиг в любую сторону никак не изменит суммы расстояний. Перемещая таким образом прямую в нужную сторону мы добьемся того, что на нее попадет по крайней мере одна точка, а сумма расстояний не увеличится.

Итак, у нас некоторая точка попала на прямую. Выполним параллельный перенос системы координат так, чтобы эта точка оказалась началом координат. Поскольку теперь наша прямая проходит через начало координат, можно все точки лежащие во второй и третьей четвертях зеркально отобразить в четвертую и первую (это не изменить расстояния от них до прямой). Пусть прямая проходит под углом φ к оси абсцисс. Тогда расстояние от точки (*xi*,*yi*) будет равно $|y_i \cos\varphi - x_i \sin\varphi|$ и суммарное расстояние от точек до прямой будет равно $d(\varphi) = (\sum k_i y_i)\cos\varphi - (\sum k_i x_i)\sin\varphi$, где $k_i = +1$, если соответствующая точка находится выше прямой, и $k_i = -1$, если ниже. На каждом интервале, не содержащем направлений на точки множества эта функция является дифференцируемой сколько угодно раз, и поэтому если бы она имела минимум внутри такого интервала, то ее производная была бы равна нулю при соответствующем значении φ, а вторая – положительна. Допустим, это выполняется для некоторой прямой. Снова выполним преобразование – повернем систему координат, чтобы эта прямая совпала с осью Ox (и не забудем перенести точки в первую или четвертую четверть). Тогда знакопеременная сумма $(\sum k_i y_i)$ будет обязательно положительной (ведь ординаты точек лежащие выше оси Ox будут браться с коэффициентом +1, а ниже – с -1). Первая производная будет равна

$$-\left(\sum k_i y_i\right)\sin\varphi - \left(\sum k_i x_i\right)\cos\varphi,$$

и поскольку$\varphi = 0$ – экстремум, следует приравнять ее в этой точке к нулю, откуда получим $\sum k_i x_i = 0$. Но тогда вторая производная

$$-\left(\sum k_i y_i\right)\cos\varphi - \left(\sum k_i x_i\right)\sin\varphi$$

в силу найденных соотношений будет отрицательной, что противоречит тому, что во внутренней точке может достигаться минимум. А значит оптимальная прямая должна проходить и через вторую точку.

Отсюда сразу же следует алгоритм для нахождения оптимальной прямой за $O(N^3)$ – перебираем все пары точек, и суммируем расстояния от всех точек до прямой, проходящей через эти две. Однако можно сократить количество действий до$O(N^2 \log N)$, если перебирать лишь вариант для одной точки. Тогда после сортировки остальных точек по углу, мы сможем за $O(N)$ вычислить все значения функций
$d(\varphi) = (\sum k_i y_i)\cos\varphi - (\sum k_i x_i)\sin\varphi$ для прямых, проходящих через каждую точку. Следует лишь при переходе к очередному направлению за $O(1)$ поправить коэффициенты $(\sum k_i y_i)$ и $(\sum k_i x_i)$.

Задача M. Дирихле и Фибоначчи

Имя входного файла:	`m.in`
Имя выходного файла:	`m.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Два великих математика Дирихле и Фибоначчи занимались разведением кроликов. У Дирихле было очень большое количество клеток, в каждой из которых могло вместиться не более k пар кроликов. Фибоначчи экспериментально вывел законы, по которым размножаются кролики. Результаты своих экспериментов он записал в таблицу $F[1:k]$, где $F[i]$ - количество новых пар кроликов, которое появится через год, если в начале года в клетку посадить i пар. Напишите программу, определяющую максимальное количество пар кроликов, которое могут вырастить за N лет эти ученые, если в начале у них была только одна пара.

Формат входного файла

В первой строке входного файла записано количество лет N ($1 \le N \le 14$), во второй - размер клетки k ($1 \le k \le 100$). Третья строка содержит элементы таблицы $F[1:k]$ ($0 \le F[i] \le 20$), отделяющиеся друг от друга пробелами.

Формат выходного файла

Единственное число - максимальное количество пар кроликов.

Примеры

m.in	m.out
2 2 1 3	5
5 4 2 4 9 13	918

Разбор задачи M. Дирихле и Фибоначчи

Очевидно, что для того, чтобы суммарное количество кроликов за N лет было максимально возможным, нужно каждый год обеспечивать максимально возможный прирост. Такую задачу можно было бы решить с помощью динамического программирования, с функцией $f(S)$ - максимальный

прирост, который могут дать S кроликов за год. Рекуррентная формула будет иметь вид $f(S) = \max f(S-i) + F[i])$. Однако S в процессе решения может очень быстро расти. И поэтому данный способ неприменим сразу. Пусть мы используем p_1 клеток размера 1, p_2 клеток размера 2 и т.д. Пусть i^* - это оптимальный размер клетки в том смысле, что $F[i^*]/i^*$ - максимально. Тогда ясно, что если у нас есть некоторое p_i, не меньшее по чем i^*, то имеет смысл преобразовать i^* клеток по i кроликов, в i клеток по i^* кроликов. Это значит, что в оптимальной рассадке кроликов по клеткам каждое p_i (кроме как для i^*) должно быть строго меньше i^*. Таким образом, наибольшее количество кроликов, для которых оптимальное распределение не содержит клеток типа i^* можно оценить величиной $S^* = (k(k+1)/2-i^*)\cdot(i^*-1)$. Отсюда получается сравнительно простой алгоритм для нахождения оптимального прироста - для первых S^* значений вычисляем значения с помощью динамического программирования, а для последующих определяем вычитаем столько раз по i^* сколько нужно для получения первого числа не превышающего S^*. Тогда добавив к оптимальному приросту для S^* соответствующее количество раз по $F[i^*]$ получим требуемый результат.

Задача N. Похожие слова

Имя входного файла:	`n.in`
Имя выходного файла:	`n.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Пусть имеется множество слов, состоящих из заглавных букв латинского алфавита. Назовем два слова *похожими*, если в них одинаковое количество букв, и они отличаются ровно в одном месте (например, STICK и STOCK). Напишите программу, которая разбивает заданное множество слов на группы слов, в каждой из которых все слова непохожи друг на друга (количество таких групп не должно превышать 26).

Формат входного файла

Во входном файле заданы слова, составляющие исходное множество - по одному в строке. Длина одного слова не превышает 26000 символов, количество слов - не более 10000. Общая длина всех слов не превышает 2600000.

Формат выходного файла

В первой строке выходного файла необходимо вывести количество N

получившихся групп слов, непохожих друг на друга (или 0, если не существует разбиения с количеством групп не более 26). Далее должны следовать N строк, в каждой из которых перечислены через пробел слова соответствующей группы.

Примеры

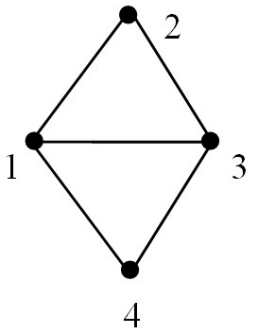
n.in	n.out
STICK SLACK SNAKE STAM STAMP STOCK FIREWORK STACK SMACKER WORK SMARK LUCKY	6 SMARK STICK WORK SMACKER STOCK STAMP LUCKY SLACK FIREWORK SNAKE STACK STAM

Разбор задачи N. Похожие слова

Ключевым для решения этой задачи является тот факт, что похожие слова не могут иметь одинаковый хэш вида $h(s) = (s_1 + s_2 + \ldots + s_{N-1} + s_N) \mod 26$. Действительно, пусть два похожих слова q и s отличаются в позиции с номером i. Тогда разность их хэшей равна $h(q) - h(s) = (q_i - s_i) \mod 26$. Если бы хэши были равны, то должны были бы быть равны и соответствующие символы q_i и s_i, что противоречит определению похожих слов. Таким образом мы каждое слово определяем в группу, номер которой определяется хэшем. Это гарантирует, что в каждой группе не будет похожих слов.

Задача O. Экстремальный граф

Имя входного файла:	`o.in`
Имя выходного файла:	`o.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб



Широко известным типом задач теории экстремальных графов является вопрос о нахождении величины $ex(v, H)$ - наибольшего количества ребер, которое может быть в графе, имеющем v вершин и не содержащем подграфа, изоморфного некоторому запрещенному графу H. (Графы H_1 и H_2 называются изоморфными, если между их множествами вершин можно установить взаимно-однозначное соответствие, при котором множества ребер этих графов переходят друг в друга). Здесь рассматриваются только неориентированные простые графы без петель. В данной задаче требуется найти $ex(v, H)$, для графа H со следующим множеством вершин и множеством ребер: V = 1, 2, 3 ,4 и E = (1,2), (2,3), (3,4), (4,1), (1,3). То есть H - это два треугольника с общим ребром. Кроме того, требуется построить сам экстремальный граф, в котором v вершин, $ex(v, H)$ ребер и нет подграфа изоморфного H.

Формат входного файла

Единственная строка входного файла содержит натуральное число $v(1 \leq N \leq 200)$.

Формат выходного файла

В первой строке выходного файла выведите число $ex(v, H)$, в последующих $ex(v, H)$ строках выведите через пробел концы ребер экстремального графа. Вершины нумеруются числами от 1 до v. Если решений несколько выведите любое из них.

Примеры

o.in	o.out
4	4 1 2 2 3 3 4 4 1

Разбор задачи O. Экстремальный граф

При $v \geq 4$ ответ $\left[v^2/4\right]$. То есть при $v = 2k$ максимальное количество ребер равно k^2, а при $v = 2k + 1$ оно равно $k^2 + k$. Пример дает полный двудольный граф с долями $[v/2]$ и $[(v+1)/2]$, так как он вообще не содержит треугольников. При $v < 4$, очевидно, что полный граф подходит. Сложность алгоритма: $O\left(v^2\right)$.

Теперь обоснуем ответ, то есть докажем, что в графе, в котором $v \geq 4$ вершин и не содержится подграфа, изоморфного H, не больше $\left[v^2/4\right]$ ребер. Докажем это утверждение индукцией по количеству вершин. Заметим, что свойство графа не содержать подграфа, изоморфного H, эквивалентно тому, что любые 4 вершины этого графа соединены не более чем четырьмя ребрами. Назовем такой граф хорошим. Ясно теперь, что база для 4-х вершин выполняется. Переход: предположим, что утверждение доказано для $v = n - 1 \geq 4$, и докажем для $v = n$. Докажем это от противного. Пусть для $v = n$ оно не верно. Если $n = 2k + 1$, то это значит, что найдется хороший граф, в котором $2k + 1$ вершина и хотя бы $k^2 + k + 1$ ребро. Так как удаление ребер не нарушает свойство графа быть хорошим, то можно считать, что ребер ровно $k^2 + k + 1$. Но тогда найдется вершина, из которой выходит не более k ребер, так как в противном случае всего ребер в графе будет не менее

$$(k+1)(2k+1)/2 = k^2 + 3k/2 + 1/2 > k^2 + k + 1$$

Удалим эту вершину, тогда в графе останется $2k$ вершин и не менее $k^2 + 1$ ребер между ними, причем граф по-прежнему хороший. Это противоречит предположению индукции. Аналогично разбирается случай $n = 2k$.

Задача P. Попарные суммы

Имя входного файла:	`p.in`
Имя выходного файла:	`p.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Мультимножеством называется такой набор чисел, в котором порядок элементов неважен, но каждое число может встречаться несколько раз. Мощностью мультимножества называется количество элементов в нём. Для данного мультимножества целых чисел $\{a_1, a_2, ..., a_n\}$ *набором попарных сумм* называется мультимножество $\{a_i + a_j\}_{1 \leq i < j \leq n}$, или в развернутом виде:

$$\{a_1 + a_2, a_1 + a_3, ..., a_1 + a_n, a_2 + a_3, ..., a_2 + a_n, ..., a_{n-1} + a_n\}.$$

Напишите программу, которая для данного n построит какие-либо два непересекающихся мультимножества мощности n с совпадающими наборами попарных сумм.

Формат входного файла

Единственная строка входного файла содержит натуральное число $n (2 \leq N \leq 4000)$.

Формат выходного файла

Выведите в выходной файл слово “Impossible”, если невозможно построить какие-либо два непересекающихся мультимножества мощности n с совпадающими наборами попарных сумм. В противном случае выведите в первой строке через пробел n чисел первого мультимножества, а во второй строке через пробел n чисел второго мультимножества. Элементы этих мультимножеств должны быть целыми числами в пределах от 0 до 100000 включительно. Если решений несколько, выведите любое из них.

Примеры

p.in	p.out
2	100 200 150 150
3	Impossible

Разбор задачи P. Попарные суммы

Лемма 1. Если n не является степенью двойки, то любое мультимножество мощности n восстанавливается по своему набору попарных сумм однозначно.

Доказательство. Пусть $A = \{a_1, a_2, ..., a_n\}$ – набор из n чисел, где $n \neq 2^k; (\forall k \geq 0)$. Проверим индукцией по p, что степенные суммы набора A

$$s_p(A) = a_1^p + a_2^p + ... + a_n^p$$

однозначно определяются значениями степенных сумм набора его попарных сумм, который мы обозначим $A^{(2)}$.

Во-первых, $s_1\left(A^{(2)}\right) = (n-1)\, s_1(A)$, поэтому

$$s_1(A) = \frac{1}{n-1} s_1\left(A^{(2)}\right).$$

Далее,

$$s_p\left(A^{(2)}\right) = \sum_{i<j}(a_i+a_j)^p = \tfrac{1}{2}\sum_{i\neq j}(a_i+a_j)^p = \tfrac{1}{2}\sum_{i\neq j}\sum_{r=0}^{p} C_p^r a_i^r a_j^{p-r} =$$
$$= \tfrac{1}{2}\sum_{r=0}^{p} C_p^r \sum_{i\neq j} a_i^r a_j^{p-r} = \tfrac{1}{2}\sum_{r=0}^{p} C_p^r \left(s_r(A)\, s_{p-r}(A) - s_p(A)\right) =$$
$$= \tfrac{1}{2}\sum_{r=1}^{p-1} C_p^r s_r(A)\, s_{p-r}(A) + s_p(A)\, s_0(A) - \tfrac{1}{2} s_p(A) \sum_{r=0}^{p} C_p^r =$$
$$= \tfrac{1}{2}\sum_{r=0}^{p-1} C_p^r s_r(A)\, s_{p-r}(A) + \left(n - 2^{p-1}\right)\cdot s_p(A).$$

В последнем равенстве мы воспользовались тем, что $s_0(A) = n$. По предположению индукции все слагаемые в первой сумме в правой части однозначно определяются степенными суммами набора $A^{(2)}$. Поэтому $s_p(A)$ при $n \neq 2^{p-1}$ тоже однозначно определяется степенными суммами набора $A^{(2)}$.

Итак, мы проверили, что степенные суммы $s_1(A)$, $s_2(A)$, ..., $s_n(A)$ однозначно определяются набором $A^{(2)}$.

Далее, обозначим $\sigma_k(A) = \sum\limits_{1\leq i_1<\ldots<i_k\leq n} a_{i_1}a_{i_2}\ldots a_{i_k}$

k-тую элементарную симметрическую функцию набора A. Тогда нетрудно проверить равенство (формула Ньютона)

$$k\sigma_k = \sigma_{k-1}s_1 - \sigma_{k-2}s_2 + \sigma_{k-3}s_3 - \ldots + (-1)^k \sigma_1 s_{k-1},$$

которая позволяет по числам $s_1(A)$, $s_2(A)$, ... последовательно находить числа $\sigma_1(A)$, $\sigma_2(A)$, ... Зная же все числа $\sigma_k(A)$, по теореме Виета мы можем найти элементы набора A как корни уравнения

$$x^n - \sigma_1 x^{n-1} + \sigma^2 x^{n-2} - \ldots + (-1)^{n-1}\sigma_{n-1}x + (-1)^n \sigma_n = 0.$$

Таким образом, набор A восстанавливается по набору $A^{(2)}$ однозначно.

Лемма 2. Пусть $k \in \mathrm{N}$ и $n = 2^k$. Пусть A – это множество всех целых неотрицательных чисел от 0 до $2n-1$, у которых сумма цифр в двоичной записи четная, аB – это множество всех целых неотрицательных чисел от 0 до $2n-1$, у которых сумма цифр в двоичной записи нечетная. Тогда каждое из множеств A и B имеет мощность n и наборы их попарных сумм совпадают.

Доказательство. Установим взаимно-однозначное соответствие между парами элементов множеств A и B. Пусть x и y какие-то два различных элемента множества A . Запишем x и y в двоичной записи: $x = \overline{x_n\ldots x_1 x_0}$, $y = \overline{y_n\ldots y_1 y_0}$. Пусть k – номер первого разряда, в котором отличаются

эти записи, то есть $xi = yi$ при $n \geq i > k$, но $x_k \neq y_k$. Ясно, что тогда одно их чисел x_k, y_k равно 0, а второе 1. Поменяем местами k-е биты чисел x y то есть рассмотрим новые числа $x' = \overline{x_n...x_{k+1}y_kx_{k-1}...x_0}$ и $y' = \overline{y_n...y_{k+1}x_ky_{k-1}...y_0}$. В силу (*) при такой замене меняется четность суммы цифр, поэтому $x', y' \in B$, и, очевидно, что $x' + y' = x + y$. Ясно, что соответствие $(x, y) \to (x', y')$ является взаимно-однозначным соответствием между множествами A и B. А так как $x + y = x' + y'$, то это означает, что наборы попарных сумм множеств A и B совпадают.

Из лемм 1 и 2 легко получается решение задачи.

Задача Q. Биномиальные коэффициенты

Имя входного файла:	`q.in`
Имя выходного файла:	`q.out`
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Биномиальными коэффициентами называются числа $C_n^k = \frac{n!}{k!(n-k)!}$. Напишите программу для вычисления остатка от деления биномиаль-ного коэффициента на степень двойки.

Формат входного файла

Входной файл содержит некоторое количество T тестов ($1 \leq T \leq 100000$). В каждой из T строк входного файла содержится три целых числа n, k и m ($1 \leq n \leq 64, 0 \leq k \leq 2^n, 1 \leq m \leq 40$), записанных через пробел.

Формат выходного файла

Для каждой тройки чисел n, k и m выведите в соответствующую строку выходного файла остаток от деления на 2^m биномиального коэффициента $C_{2^n-1}^k$.

Примеры

q.in	q.out
1 0 30 1 1 20 3 6 2	1 1 3

Разбор задачи Q. Биномиальные коэффициенты

Для данного натурального m и нечетного j будем обозначать через $inv_m(j)$ решение сравнения $xj \equiv 1 (mod 2^m)$. Оно может найдено либо посредством расширенного алгоритма Евклида, либо индукцией по m. Ключевым для получения эффективного алгоритма нахождения требуемого остатка является сравнение (2). Получим его. Имеем при $n < m \leq 2n$:

$$C_{2^n-1}^k = \prod_{j=1}^{k} \frac{2^n-j}{j} = \prod_{j=1}^{[k/2]} \frac{2^n-2j}{2j} \cdot \prod_{j=1}^{[(k+1)/2]} \left(\frac{2^n}{2j-1} - 1\right)$$
$$\equiv C_{2^{n-1}-1}^{[k/2]} \cdot (-1)^{[(k+1)/2]} \cdot \prod_{j=1}^{[(k+1)/2]} (1 - 2^n inv_m(2j-1)) (mod 2^m)$$

Так как $m \leq 2n$, то при раскрытии скобок в произведении $\prod_{j=1}^{[(k+1)/2]} (1 - 2^n inv_m(2j-1))$ все члены, кроме слагаемых в сумме $1 - 2^n \sum_{j=1}^{[(k+1)/2]} inv_m(2j-1)$, будут кратны 2^{2n}, а значит и 2^m. Поэтому

$$C_{2^n-1}^k \equiv C_{2^{n-1}-1}^{[k/2]} \cdot (-1)^{[(k+1)/2]} \cdot \left(1 - 2^n \sum_{j=1}^{[(k+1)/2]} inv_m(2j-1)\right)$$
$$\equiv C_{2^{n-1}-1}^{[k/2]} \cdot (-1)^{[(k+1)/2]} \cdot \left(1 - 2^n \sum_{j=1}^{[(k+1)/2]} inv_{m-n}(2j-1)\right) (mod 2^m) \quad (1)$$

Последний переход осуществлен благодаря множителю 2^n перед суммой.

Обозначим через $S_m(k)$ сумму $\sum_{j=1}^{[(k+1)/2]} inv_m(2j-1)$. Ясно, что $S_1(k) = [(k+1)/2]$.

При $m > 1$, так как $inv_m(2j-1) + inv_m(2^m - 2j + 1) \vdots 2^m$, то $S_m(2^m) \vdots 2^m$. Поэтому $S_m(k) \equiv S_m(k\%2^m) (mod 2^m)$, где % означает взятие остатка по модулю.

Сравнение (1) в новых обозначениях примет вид:

$$C_{2^n-1}^k \equiv C_{2^{n-1}-1}^{[k/2]} \cdot (-1)^{[(k+1)/2]} \cdot (1 - 2^n S_{m-n}(k)) (mod 2^m),\ n < m \leq 2n. \quad (2)$$

При $m \leq n$ сравнение (2) примет совсем простой вид

$$C_{2^n-1}^k \equiv C_{2^{n-1}-1}^{[k/2]} \cdot (-1)^{[(k+1)/2]} (mod 2^m),\ m \leq n \quad (2')$$

Итеируя формулу (2), то есть, применяя ее к биномиальному коэффициенту $C_{2^{n-1}-1}^{[k/2]}$ в правой части, определенное число раз, мы можем све-

сти вычисление $C_{2^n-1}^k \% 2^m$ к вычислению таких величин: $C_{2^{p-1}-1}^{\left[k/2^{n-p+1}\right]} \% 2^m$, $S_{m-n}(k)$, $S_{m-n-1}([k/2])$,. . . ,$S_{m-p}([k/2^{n-p}])$ для некоторого $p \leq n$ такого, что $m \leq 2p$. Так как $m \leq 40$, то можно выбрать $p \leq 20$.

Теперь о реализации. Преподсчитаем все числа $inv_{40}(2j-1)$, для $1 \leq j \leq 2^{19}$, и $S_{40}(k)$ при $1 \leq k \leq 2^{20}$. Также преподсчитаем остатки $C_{2^{n-1}}^k \% 40$ для $1 \leq n \leq 19$ и $0 \leq k < 2^n$, здесь понадобится таблица чисел $inv_{40}(2j-1)$. Так как $S_m(k)$ сводится к $S_m(l)$ для некоторого $l < 2^m$, то этой информации хватит, чтобы по описанному выше методу быстро посчитать искомый остаток, а именно за $O(n)$.

Задача R. Биномиальные коэффициенты 2

Имя входного файла:	`r.in`
Имя выходного файла:	`r.out`
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

В этой задаче вам придется выполнить задание обратное к тому, которое предлагалось в предыдущей задаче.

Напишите программу для определения аргументов биномиального коэффициента, при которых он имеет заданный остаток от деления на степень двойки.

Для каждой пары чисел n, r входного файла выведите в отдельной строке выходного файла количество тех чисел k ($0 \leq k \leq 2^n$), для которых биномиальный коэффициент $C_{2^n-1}^k$ дает остаток r при делении на 2^n, а затем выведите сами эти числа в порядке возрастания через пробел в этой же строке.

Формат входного файла

Входной файл содержит некоторое количество T тестов ($1 \leq T \leq 100000$). В каждой из T строк входного файла содержится два целых числа n и r ($1 \leq n \leq 40, 0 \leq r \leq 2^n$), записанных через пробел.

Формат выходного файла

Для каждой пары чисел n, r выведите в соответствующую строку выходного файла количество таких чисел k ($0 \leq k \leq 2^n$), для которых биномиальный коэффициент $C_{2^n-1}^k$ дает остаток r при делении на 2^n, а затем сами эти числа в порядке возрастания.

Примеры

r.in	r.out
1 0 1 1 3 7	0 2 0 1 2 1 6

Разбор задачи R. Биномиальные коэффициенты 2

Для $n \in \mathrm{Z}_+$ обозначим через $S_2(n)$ сумму цифр в двоичной записи числа n. Также для $n \in \mathrm{N}$ обозначим через $\deg_2(n)$ показатель максимальной степени двойки, на которую делится n. Как известно,

$$\deg_2(n!) = \left[\frac{n}{2}\right] + \left[\frac{n}{4}\right] + ... + \left[\frac{n}{2^j}\right] + ... = n - S_2(n).$$

Поэтому из определения C_n^k следует, что

$$\deg_2\left(C_n^k\right) = S_2(k) + S_2(n-k) - S_2(n).$$

Лемма 1. Все числа $C_{2^n-1}^k$ при $n \in \mathrm{N}$ и $0 \le k < 2^n$ являются нечетными.

Доказательство. Надо доказать, что $\deg_2\left(C_{2^n-1}^k\right) = 0$.

Запишем k в двоичной записи: $k = \overline{k_{n-1}...k_1k_0}$.

Тогда $2^n - 1 - k = \underbrace{\overline{1...11}}_{n} - \overline{k_{n-1}...k_1k_0} = \overline{(1-k_{n-1})...(1-k_1)(1-k_0)}$.

Поэтому $\begin{array}{l} S_2(2^n-1-k) = (1-k_{n-1}) + ... + (1-k_1) + (1-k_0) = \\ = n - (k_{n-1} + ... + k_1 + k_0) = n - S_2(k). \end{array}$

Откуда следует, что $\begin{array}{l} \deg_2\left(C_{2^n-1}^k\right) = S_2(k) + S(2^n-1-k) - S_2(2^n-1) = \\ = S_2(k) + n - S_2(k) - n = 0. \end{array}$

Лемма 2. Для $n \ge 2$ и $0 \le s < 2^{n-1}$ справедливо сравнение $C_{2^n-1}^{2s} + C_{2^n-1}^{2s+1} \equiv 2^n \left(mod 2^{n+1}\right)$.

Доказательство. Надо доказать, что $\deg_2\left(C_{2^n-1}^{2s} + C_{2^n-1}^{2s+1}\right) = n$.

По свойствам биномиальных коэффициентов $C_{2^n-1}^{2s} + C_{2^n-1}^{2s+1} = C_{2^n}^{2s+1}$.

Запишем $2s+1$ в двоичной записи: $2s+1 = \overline{s_{n-1}...s_1 1}$.

Тогда $2^n - (2s+1) = (2^n-1) - 2s = \underbrace{\overline{1...11}}_{n} - \overline{s_{n-1}...s_1 0} =$

$= \overline{(1-s_{n-1})...(1-s_1)\,1}$

Поэтому $S_2\left(2^n-(2s+1)\right)=(1-s_{n-1})+...+(1-s_1)+1=$
$=n+1-(s_{n-1}+...+s_1+1)=n+1-S_2\left(2s+1\right).$

Откуда следует, что $\deg_2\left(C_{2^n}^{2s+1}\right)=S_2\left(2s+1\right)+S\left(2^n-(2s+1)\right)-S_2\left(2^n\right)=$
$=S_2\left(2s+1\right)+n+1-S_2\left(2s+1\right)-1=n.$

В поисках тех чисел k, для которых $C_{2^n-1}^k$ дает остаток r при делении на 2^n можно ограничиться лишь числами $k<2^{n-1}$, так как $C_{2^n-1}^k=C_{2^n-1}^{2^n-1-k}$. Таких k всего 2^{n-1}, столько же, сколько нечетных остатков при делении на 2^n, а по лемме 1 все биномиальные коэффициенты $C_{2^n-1}^k$ нечетные. Докажем, что для каждого нечетного r от 0 до 2^n-1 найдется хотя бы одно подходящее k от 0 до $2^{n-1}-1$. Из этого по замечанию выше будет следовать, что такое k для каждого r единственно. Будем доказывать это индукцией по n. Переходя от остатков к сравнениям, мы можем считать r любым целым нечетным числом. База для $n=1$ и $n=2$ очевидна. Пусть $n\geq 3$ и уже доказано, что для каждого целого нечетного r найдется k от 0 до $2^{n-2}-1$ такое, что выполнено сравнение $C_{2^{n-1}-1}^k\equiv r\left(mod 2^{n-1}\right)$. Заметим, что из формулы (2') решения задачи "Биномиальные коэффициенты" следует, что при $0\leq x<2^{n-2}$ и $y\in\{0,1\}$ выполнено сравнение

$C_{2^n-1}^{2x+y}\equiv(-1)^{x+y}\cdot C_{2^{n-1}-1}^x\left(mod 2^n\right)$ (1)

Пусть (1) теперь фиксировано целое нечетное r. По предположению индукции найдется такое x от 0 до $2^{n-2}-1$, что

$C_{2^{n-1}-1}^x\equiv r\left(mod 2^{n-1}\right)$ (2).

Если $C_{2^{n-1}-1}^x\equiv r\left(mod 2^n\right)$ (это можно быстро проверить, используя алгоритм из задачи "Биномиальные коэффициенты"), то беря y той же четности, что и x, получим в силу формулы (1), что для $k=2x+y$ выполнено сравнение $C_{2^n-1}^k\equiv(-1)^{x+y}r\equiv r\left(mod 2^n\right)$, то есть нужное k найдено и оно лежит в пределах от 0 до $2^{n-1}-1$. Если же сравнение $C_{2^{n-1}-1}^x\equiv r\left(mod 2^n\right)$ не выполнено, то в силу (2) выполнено сравнение $C_{2^{n-1}-1}^x\equiv r+2^{n-1}\left(mod 2^n\right)$. Рассмотрим число $z=x+(-1)^x$. Ясно, что числа x и z – соседние целые числа, причем меньшее из них всегда четное, поэтому, во-первых, $0\leq z\leq 2^{n-2}-1$ (так как $n\geq 3$), а во-вторых, по лемме 2 выполнено сравнение

$$C_{2^{n-1}-1}^z\equiv 2^{n-1}-C_{2^{n-1}-1}^x\equiv 2^{n-1}-r-2^{n-1}\equiv -r\left(mod 2^n\right).$$

Значит, беря $y\in\{0,1\}$ так, чтобы числа y и z были разной четности, получим в силу формулы (1), что для $k=2z+y$ выполнено сравнение $C_{2^n-1}^k\equiv(-1)^{z+y}\cdot(-r)\equiv r\left(mod 2^n\right)$, то есть нужное k опять найдено и оно снова лежит в пределах от 0 до $2^{n-1}-1$.

Таким образом, при четном r количество искомых чисел k равно нулю, а при нечетном – двум, и они имеют вид k и 2^n-1-k, где k может быть

найдено за $O(n)$ с помощью индуктивного доказательства, изложенного выше.

Задача S. Пирамида из чисел

Имя входного файла:	`s.in`
Имя выходного файла:	`s.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Древние египтяне славились умением складывать пирамиды из каменных блоков. Мы же попробуем составить пирамиду из чисел. Пусть нам заданы 2^N целых чисел. Мы их выпишем в каком-нибудь порядке в строку. Затем над ними выше в следующей строке между каждой парой чисел запишем их произведение. Тем самым у нас получится на одно число меньше, чем в предыдущей строке. В следующую строку опять же выписываются попарные произведения. И так продолжается до тех пор, пока мы не дойдем до вершины пирамиды, где будет находится одно число. Таким образом, единственное чем мы можем распоряжаться при построении пирамиды - это порядок чисел в самой нижней строке. Напишите программу для определения того, в каком порядке следует записывать числа, чтобы на вершине пирамиды получилось максимально возможное число.

Формат входного файла

В первой строке входного файла задается число N ($1 \leq N \leq 16$), а во второй - 2^N чисел, не превышающих 10^9 по абсолютной величине.

Формат выходного файла

В входной файл выведите заданные числа в том порядке, который обеспечит максимум на вершине пирамиде.

Примеры

s.in	s.out
2 3 4 5 1	3 5 4 1
3 2 2 2 2 2 2 2 2	2 2 2 2 2 2 2

Разбор задачи S. Пирамида из чисел

Нетрудно заметить, что показатели степеней, с которыми числа нижней строки войдут в число, записанное на вершине, равны биномиальным коэффициентам. Ясно, что большему показателю должно соответствовать большее число. Таким образом, в середину мы поставим два максимума, рядом с ними - следующие по величине числа и т.д. По краям окажутся два минимальных элемента. Однако в задаче мы имеем дело не обязательно с положительными числами. Если встречается хотя бы один нулевой элемент, то подойдет любая перестановка - результат все равно будет 0. Задача резко усложняется, когда начинаем рассматривать отрицательные числа. Тогда на верху может оказаться и отрицательное число, а его нам желательно сделать минимальным по модулю. К счастью, когда количество чисел на равно степени двойки - мы имеем дело с коэффициентами $C_{2^n-1}^k$, которые по лемме 1 из предыдущей задачи все являются нечетными. Поэтому знак числа на верху определяется не расстановкой, а лишь количеством отрицательных чисел в последовательности. Если их - четное число, то на вершине будет положительное число и действуем как и прежде (учитывая лишь, что сравнения нужно производить по модулю), а если - нечетное, то необходимо минимизировать модуль верхнего число (оно будет отрицательным). Для этого в середину мы поставим два минимальных по модулю числа, рядом - с ними следующие по величине, по краям будут максимумы.

Задача T. k-суммы с повторениями

Имя входного файла:	`t.in`
Имя выходного файла:	`t.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Мультимножеством называется такой набор чисел, в котором порядок элементов неважен, но каждое число может встречаться несколько раз. Мощностью мультимножества называется количество элементов в нём. Для данного мультимножества натуральных чисел $\{a_1, a_2, ..., a_n\}$ *набором k-сумм с повторениями* называется мультимножество $\{a_{i_1} + a_{i_2} + ... + a_{i_k}\}_{1 \le i_1 \le i_2 \le ... \le i_k \le n}$. Обозначим через $f(n,k)$ мощность этого мультимножества.

Требуется восстановить по набору k-сумм с повторениями исходный набор чисел.

Формат входного файла

В первой строке входного файла содержится два натуральных числа n и k. Они удовлетворяют неравенствам $k \leq 100000$, $f(n,k) \leq 100000$. Во второй строке через пробел записаны через пробел $f(n,k)$ чисел набора k-сумм с повторениями некоторого мультимножества натуральных чисел мощности n. Все числа во второй строке входного файла не превосходят 109 и записаны в произвольном порядке.

Формат выходного файла

Выведите в выходной файл через пробел числа исходного набора, упорядоченные по возрастанию. Гарантируется, что хотя бы одно решение существует. Если решений несколько выведите любое из них.

Примеры

t.in	t.out
1 1 1 1	11
2 2 2 3 4	1 2

Разбор задачи T. k-суммы с повторениями

Пусть $A = \{a_1, a_2, ..., a_n\}$ некоторый набор натуральных чисел, упорядоченный по возрастанию: $a_1 \leq a_2 \leq ... \leq a_n$.

Обозначим через $A(k)$ набор k-сумм с повторениями набора A.

Покажем, как по набору $A(k)$ восстановить набор A. Ясно, что наименьший элемент набора $A(k)$ равно ka_1. Поэтому a_1 мы можем найти. Наименьший среди оставшихся элементов набора $A(k)$ равен $(k-1)a_1 + a_2$. Так как мы знаем a_1, то и a_2 мы можем найти. Удалим из набора $A(k)$ все k-суммы, построенные на основе числе a_1 и a_2, то есть числа $ja_1 + (k-j)a_2$, $0 \leq j \leq k$.

Наименьшим среди оставшихся чисел является $(k-1)a_1 + a_3$. Значит, и a_3 мы можем найти.

И вообще, если мы уже нашли числа $a_1, a_2, ..., a_{i-1}$ и из набора $A(k)$ удалены все k-суммы, построенные на основе этих чисел, то есть, удалены все числа $a_{j_1} + a_{j_2} + ... + a_{j_k}$, $1 \leq j_1 \leq j_2 \leq ... \leq j_k \leq i-1$,
то наименьшим среди оставшихся чисел набора $A(k)$ является $(k-1)a_1 + a_i$. Значит, мы можем найти a_i. После чего (чтобы найти a_{i+1}) надо удалить из набора $A(k)$ все еще неудаленные k-суммы, построенные на основе чисел $a_1, a_2, ..., a_i$, то есть числа $a_i + a_{j_1} + ... + a_{j_{k-1}}$, $1 \leq j_1 \leq ... \leq j_{k-1} \leq i$.

В итоге мы найдем все числа набора A. Причем, как видно из доказательства, набор A однозначно восстанавливается по набору $A(k)$.

Теперь о реализации. Набор $A\,(k)$ и действия над ним можно моделировать двумя способами, либо с помощью шаблона multiset библиотеки set в STL, либо с помощью двух массивов. Первый массив содержит все различные значения элементов набора $A\,(k)$ в порядке возрастания, а второй – для каждого значения первого массива содержит количество элементов набора $A\,(k)$ равных этому значению. Тогда поиск удаляемого элемента можно делать с помощью двоичного поиска в первом массиве, а чтобы удалить элемент надо просто уменьшить соответствующий элемент второго массива.

День второй (19.02.2009г). Контест Михаила Дворкина

Об авторе...

Дворкин Михаил Эдуардович, студент 5-го курса СПбГУ ИТМО.

Основное место работы: учитель программирования в Лицее «Физико-техническая школа».

Золотой медалист Чемпионата мира по программированию ACM ICPC 2007 года.

Член жюри и автор задач четверть- и полуфинала ACM ICPC, член Научного комитета Всероссийской олимпиады школьников по информатике.

На соревнованиях TopCoder и в Живом Журнале пишет под псевдонимом darnley.

Теоретический материал. Конечные автоматы и регулярные выражения

В данной лекции рассматриваются четыре способа задания языка:

- Детерминированный конечный автомат (ДКА).
- Недетерминированный конечный автомат (НКА).
- Нетерминированный конечный автомат с ε-переходами (ε-НКА).
- Регулярное выражение (РВ).

Эти четыре способа (как будет показано ниже) эквивалентны, то есть язык, заданный одним из этих способов, может быть задан любым другим. Класс языков, задаваемых этими способами, называется классом регулярных языков.

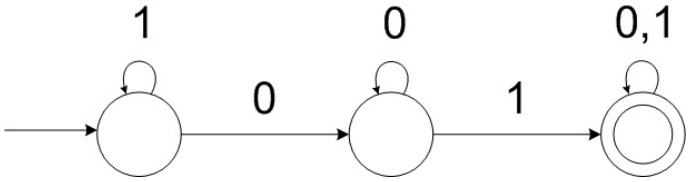
Детерминированный конечный автомат (ДКА)

Детерминированным конечным автоматом называется пятерка $(Q,\ \Sigma,\ \delta,\ q_0,\ F)$, где Q — конечное множество состояний, Σ — алфавит, над которым строится язык, δ: $Q \times \Sigma \to Q$ - функция переходов, q_0 — начальное состояние, F — множество терминальных состояний.

Обработка строки s детерминированным конечным автоматом A происходит следующим образом. Говорят, что в начале ДКА находится в состоянии q_0 и имеет на входе строку s. Затем, каждый раз автомат считывает и удаляет первый символ с из строки s и переходит в состояние $\delta(q, c)$, где q — текущее состояние автомата. Когда входная строка исчерпана, автомат находится в некотором состоянии. Если оно терминальное, то говорят, что автомат принимает строку s, иначе — не принимает.

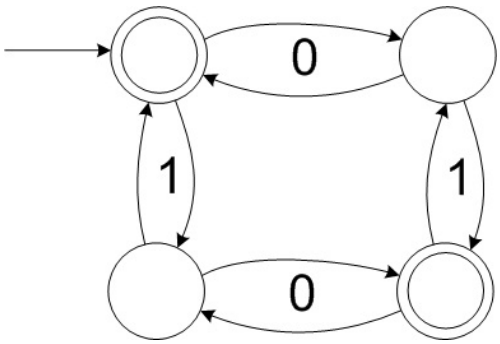
Для изображения автоматов используют также графическую нотацию.

Рассмотрим пример:



Данный автомат принимает слова, содержащие подстроку “01”. Начальное состояние обозначено стрелкой, ведущей “из ниоткуда”. Терминальные состояния (в данном случае одно) — вложенными кружками. Переходы – стрелками, помеченными символами из алфавита.

Следующий автомат принимает слова, содержащие либо нечётное число 0 и нечётное число 1, либо четное число 0 и четное число 1.



Легко понять, что этот язык может быть описан проще — это язык слов четной длины (убедитесь в этом). Возникает естественный вопрос: каков наиболее простой автомат, принимающий данный язык. Оказывается, любой детерминированный конечный автомат может быть минимизирован, причем результат минимизации единственный (с точностью до переименования вершин).

Для этого введем определение: два состояния p и q называются различимыми, если существует такая строка s, что автомат, начиная из p,

обработав строку s, перейдет в терминальное состояние, а начиная из q — в нетерминальное, либо наоборот.

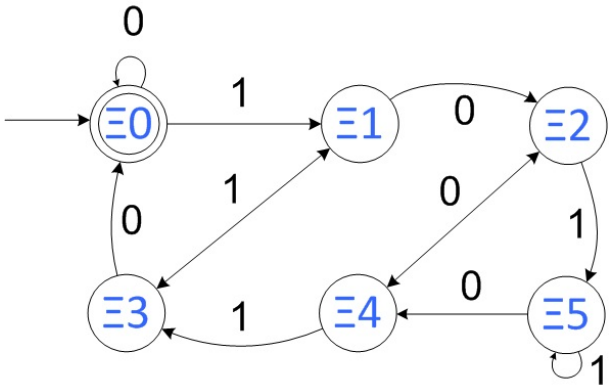
Тогда сразу видны некоторые пары различимых состояний: если p — терминальное, а q — нетерминальное, то пара (p, q) — различима, так как эти два состояние различает пустое слово.

Затем, пусть пара (p, q) различима, и из состояния r ведет по символу c ребро в p, а из состояния t по символу c — в q. Тогда пара (r, t) также различима.

Этих двух правил достаточно, чтобы найти все пары различимых состояний. Начав с пар вида (терминальная, нетерминальная), следует применить любой обход графа (в глубину или в ширину).

Так мы узнаем все пары различимых состояний. А значит, и все пары неразличимых состояний. Причем отношение “два состояния неразличимы” является отношением эквивалентности, а значит, можно рассмотреть классы эквивалентности по этому отношению. Эти классы эквивалентности соответствуют состояниям минимизированного автомата.

Рассмотрим пример. Приведенный ниже автомат A принимает язык двоичных чисел, делящихся на 6.



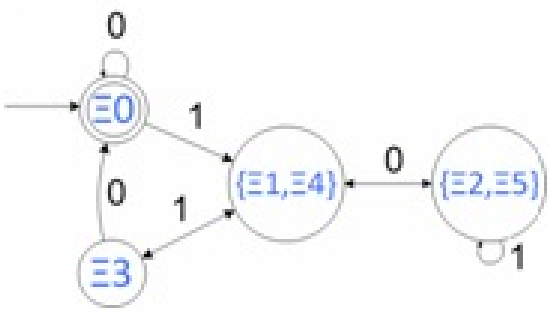
Сразу (в силу терминальности и нетерминальности) видны пары различимых состояний (≡0, ≡1), (≡0, ≡2), (≡0, ≡3), (≡0, ≡4) и (≡0, ≡5). От этих пар надо идти по обратным ребрам. Например, рассмотрим различимую пару (≡0, ≡2). В состояние ≡0 ведут по символу “0” ребра из состояний ≡0 и ≡3. В состояние ≡2 по символу “0” — из состояний ≡1 и ≡4. А значит, различимы следующие пары состояний: (≡0, ≡1), (≡0, ≡4),

($\equiv$3, $\equiv$1) и ($\equiv$3, $\equiv$4). Помечаем эти пары как различимые (две из них уже были помечены ранее) и продолжаем обход всех пар.

В итоге получим следующую таблицу:

	≡0	≡1	≡2	≡3	≡4	≡5
≡0		≠	≠	≠	≠	≠
≡1	≠		≠	≠		≠
≡2	≠	≠		≠	≠	
≡3	≠	≠	≠		≠	≠
≡4	≠		≠	≠		≠
≡5	≠	≠		≠	≠	

А значит, классы эквивалентности таковы: $\equiv$0, $\equiv$1, $\equiv$4, $\equiv$2, $\equiv$5, $\equiv$3. И искомый минимизированный автомат выглядит так:

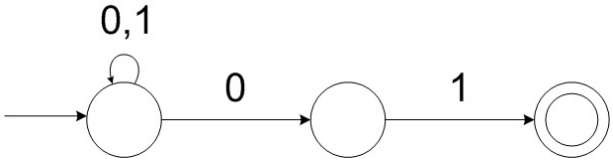


Недетерминированный конечный автомат (НКА)

Недетерминированный конечный автомат отличается тем, что из одного состояния по одному символу может быть не один переход (именно столько было в случае ДКА), а ноль или несколько.

Автомат, принимающий слово, каждый раз может “угадать”, по которому из возможных переходов следует идти. Говорят, что автомат принимает слово s, если существует такая последовательность переходов (такая последовательность “угадываний”), которая переводит автомат в терминальное состояние.

Рассмотрим в качестве примера следующий автомат. Он принимает слова, заканчивающиеся на “01”.



Как же он работает? Пусть ему подается строка “0001”. Считывая первые два нуля, автомат остается в начальном состоянии. Считывая третий ноль, автомат угадывает, что этот ноль является предпоследним и переходит в центральное состояние, где считывает единицу, переходит в терминальное состояние и принимает данное слово.

Таким образом, в каждый момент времени есть не одно состояние, а целое множество состояний, в которых может находиться автомат. В начале — это множество из одного начального состояния. Когда автомат считывает символ c, он может перейти по любому переходу из любого состояния, в котором мог находиться. Таким образом, новое множество возможных состояний — это множество состояний, в которые ведет хотя бы одно ребро по символу c из текущего множества возможных состояний.

Соответственно, можно построить ДКА, эквивалентный данному НКА. Состояние ДКА — множество состояний НКА. Начальное состояние — множество из одного начального состояния НКА. Терминальные состояния — множества, которые содержат хотя бы одно терминальное состояние НКА.

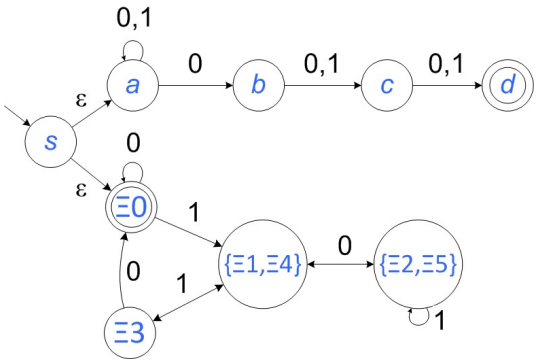
Следует понимать, что построенный таким образом автомат экспоненциально больше оригинального НКА. Однако, с теоретической точки зрения, мы видим, что ДКА и НКА — “равномощные” способы описания языков.

Недетерминированный конечный автомат с ε-переходами (ε-НКА)

Иногда для простоты проектирования в НКА добавляются ребра, помеченные не символом, а пустым словом - ε. По такому переходу автомат может перейти в любой момент времени, не считывая символа из строки, подаваемой ему на вход.

В частности, если ε-НКА в данный момент может находиться в состоянии q, то он может находиться и в любом состоянии, достижимом из q

по одному или нескольким ε-переходам. Множество состояний, в которое можно перейти по ε-переходам из q называется ε-замыканием состояния q. Рассмотрим пример ε-НКА. Данный автомат принимает слова, в которых предпредпоследний символ – ноль, а также двоичные записи чисел, делящихся на 6.



В нем ε-переходы позволяют в самом начале “спонтанно” выбрать, чего же ожидать — слова с предпредпоследним нулем или числа, кратного 6.

Рассмотрим ДКА, эквивалентный этому ε-НКА. Состояния ДКА — снова множества состояний ε-НКА (множество состояний, в котором может в данный момент времени находиться ε-НКА). Начальное состояние на этот раз не $\{s\}$, а $\{s, a\ , \equiv 0\}$, поскольку ε-НКА мог из s сразу же перейти по ε-переходу. Терминальные состояние — множества состояний, содержащие хотя бы одно терминальное.

Регулярные выражения

Регулярное выражение — способ описать язык одной строкой.
Если R — регулярное выражение, то $L(R)$ — язык, который оно задает. Перечислим все допустимые конструкции:

- “a”, “b”,. . . , “z” — задает язык, состоящий из одного слова — соответствующей буквы;
- “$\emptyset$” — задает язык $\emptyset$ (пустое множество);
- “ε” — задает язык ε, состоящий из одного слова — пустого;
- (x) — задает тот же язык, что и регулярное выражение x;
- x^* — задает замыкание Клини языка, задаваемого регулярным выражением x;

- xy — задает конкатенацию языков, задаваемых регулярными выражениями x и y;
- $x|y$ — задает язык, состоящий из объединения языков, задаваемых x и y.

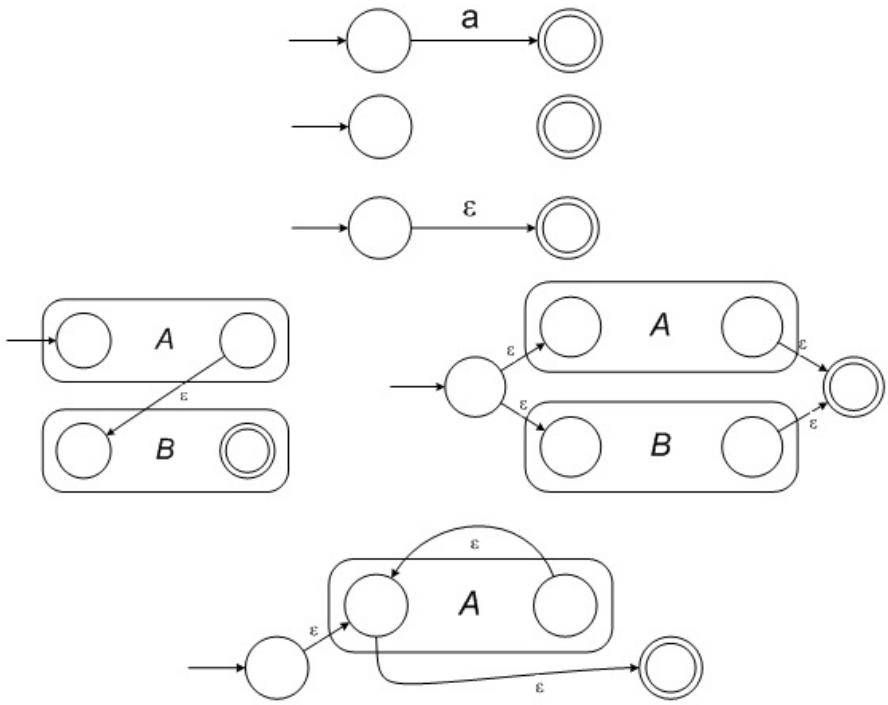
Рассмотрим язык L. Замыканием Клини языка L называется язык L^*, состоящий из слов вида $s_1 s_2 \ldots s_n$, где $n \geq 0$, и все слова s_i принадлежат языку L. Например 1^* — язык слов, состоящих только из единиц.

Конкатенация двух языков L и M — язык слов вида st, где s — слово из L, а t — слово из M. Рассмотрим несколько примеров.

$(0|1)^*$ — язык всех слов над алфавитом {“0”, “1”}.
$(0|11)^*$ — язык слов, в которых все группы подряд идущих единиц имеют четную длину.
$(0|1)^*(00|11)(0|1)^*$ — язык слов, в которых встречаются две одинаковые цифры подряд.

Оказывается, регулярные выражения — способ задавать языки столь же мощный, сколь и автоматы.

Для любого регулярного выражения существует ε-НКА, принимающий тот же язык, что и это РВ. Этот ε-НКА строится рекурсивно. Ниже приведены конструкции, позволяющие выразить с помощью ε-НКА все возможные конструкции, используемые в регулярных выражениях.



Наоборот, по автомату A (а именно ДКА) можно построить эквивалентное ему регулярное выражение. Для этого строится трехмерная таблица регулярных выражений: R_{ijk} — регулярное выражение, задающее слова, переводящие автомат A из состояния i в состояние j, причем так, что промежуточные состояния имеют номера не более k.

Так, $R_{ii0} = a|b|\varepsilon$, где “a” и “b” — символы, написанные на ребрах, ведущих из состояния i в себя (это слова, переводящие автомат из состояния i в себя, вообще не используя промежуточные состояния).

Далее, $R_{ij0} = a|b$, где “a” и “b” — символы, написанные на ребрах, ведущих из i в j. Если же таких ребер нет, то $R_{ij0} = \emptyset$.

Наконец, рассмотрим общий случай R_{ijk}. Как можно перевести автомат из состояния i в состояние j, используя промежуточные состояния с 1-го по k-е? Первый вариант: вообще не заходя в состояние k. Такие способы описываются выражением $R_i,j,k-1$. Второй вариант: состояние номер k посещается. Тогда путь — из состояния i в состояние k с промежуточными состояниями, меньшими k; затем несколько раз (возможно, ноль) — из состояния k в состояние k с промежуточными состояниями, меньшими k; наконец, из состояния k в состояние j с промежуточными состояниями, меньшими k.

Имеем следующее рекуррентное соотношение:
$R_{ijk} = R_{i,j,k-1} | R_{i,k,k-1} R_{k,k,k-1} * R_{k,j,k-1}$.

Наконец, регулярное выражение, эквивалентное всему автомату A — это объединение выражений $R_{q0,t,|Q|}$ по всем терминальным состояниям t.

Выражение, получается невероятно громоздким, однако нами теоретически доказан факт, что автоматы эквивалентны регулярным выражениям. Интересно, что синтаксис регулярных выражений можно дополнять полезными конструкциями, которые не увеличивают класс языков, которые можно задать. Вот некоторые из них:

- $x \cap y$ — пересечение языков $L(x)$ и $L(y)$;
- $\bar{x}$ дополнение языка $L(x)$ — язык всех слов, не принадлежащих $L(x)$;
- $x\ y$ — разность языков $L(x)$ и $L(y)$ — язык слов принадлежащих $L(x)$ и не принадлежащих $L(y)$;
- x^R — обращение языка$L(x)$ — язык, состоящий из слов языка $L(x)$, прочитанных слева направо.

Языки, которые можно задать одним из описанных четырех способов (а значит, и тремя другими) называются регулярными.

Заключение

Данный материал изложен чрезвычайно подробно и понятно в книге Д. Хопкрофта, Р. Мотвани и Д. Ульмана “Введение в теорию автоматов, языков и вычислений”.

Автор благодарит своего преподавателя Андрея Сергеевича Станкевича, который интересно и качественно читает соответствующий курс в Санкт-Петербургском государственном университете информационных технологий, механики и оптики.

Задачи и разборы

Задача A. Минимизация ДКА

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	10 с
Ограничение по памяти:	256 Мб

Дан детерминированный конечный автомат A.

Постройте детерминированный конечный автомат, принимающий тот же язык, что и A, и имеющий наименьшее возможное число состояний.

Формат входного файла

Первая строка входного файла содержит алфавит Σ, который является непустым подмножеством латинского алфавита (все буквы строчные).

Следующая строка содержит число $|Q|$ - количество состояний автомата ($1 \leq |Q| \leq 50$). Состояния нумеруются числами от 1 до $|Q|$. Следующая строка содержит число q_0 ($1 \leq q_0 \leq Q$) — номер начального состояния, затем число $|F|$ — количество терминальных состояний, затем $|F|$ чисел от 1 до $|Q|$ — номера терминальных состояний. Следующие $|Q|$ строк содержат по $|\Sigma|$ чисел — описание функции δ.

Формат выходного файла

Выведите описание искомого детерминированного конечного автомата в формате, описанном выше, но без первой строки (строки с алфавитом).

Пример

a.in	a.out
ab 5 1 2 2 3 2 3 1 4 4 1 3 2 5 5	2 1 1 2 2 2 1 1

Данный автомат принимает слова нечетной длины из букв “a” и “b”.

Задача B. Обращение ДКА

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Дан детерминированный конечный автомат A. Пусть L – язык, который он принимает. Постройте детерминированный конечный автомат, принимающий язык LR – язык, состоящий из слов языка L, прочитанных справа налево.

Формат входного файла

Входной файл содержит описание автомата A в формате, описанном в задаче "A". В данной задаче $|Q| \leq 10$.

Формат выходного файла

Выведите описание искомого детерминированного конечного автомата в аналогичном формате, но без первой строки (строки с алфавитом). Число состояний в построенном автомате не обязано быть минимальным, но не должно превышать 20 000.

Пример

b.in	b.out
ab 5 1 1 4 2 2 3 3 4 5 4 4 5 5	8 1 4 2 4 6 8 5 1 5 1 6 2 6 2 7 3 7 3 8 4 8 4 2

Данный во входном файле автомат принимает слова из букв "a" и "b", у которых третий символ – "a". В выходной файл следует вывести описание автомата, принимающего обращенный язык: множество слов, у которых третий **с конца** символ – "a".

Задача C. ДКА, проверяющий делимость

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

В условиях данной задачи разрешим в двоичной записи чисел присутствие ведущих нулей. Кроме того разрешим записывать число 0 в двоичной записи как ε (как пустую строку).

Таким образом, строки “101” и “000101” являются двоичными записями числа 5, а строки “000000” и “” – двоичными записями числа 0. Дано три различных положительных целых чисел a, b и c.

Постройте ДКА над алфавитом “0”, “1”, принимающий те и только те строки, которые являются двоичными записями чисел, делящихся на хотя бы одно число из $\{a, b, c\}$.

Число состояний в построенном автомате не обязано быть минимальным, но не должно превышать 20 000. Гарантируется, что существует искомый ДКА с не более чем 1 000 состояний.

Формат входного файла

Во входном файле содержится три различных положительных нечетных числа a, b, c. ($1 \leq$a,b,c< 1000).

Формат выходного файла

Выведите описание искомого автомата в формате, описанном в задаче “A”, но без первой строки (строки с алфавитом).

Пример

c.in	c.out
3 3 5 15	15 15 7 3 5 6 9 10 12 15 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1

Задача D. Укладка плиток

Имя входного файла:	d.in
Имя выходного файла:	d.out
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Фирма “Автоматическая смена пола” разработала робота, который замощает плиткой дорожки. У робота есть неограниченный запас плиток размером 2×1. Дорожка, которую надо замостить имеет размеры $m \times n$, то есть состоит из $m \cdot n$ клеток. Здесь m – ширина дорожки, n – ее длина. Дорожка идет с запада на восток.

Программа для робота – это последовательность команд “H”, “V” и “S”. Робот начинает выполнять ее, стоя на северо-западной клетке дорожки. Получив команду “H”, робот кладет очередную плитку горизонтально – длинной стороной с запада на восток, так что ее западная половина совпадает с клеткой, на которой робот стоит в данный момент.

Получив команду “V”, робот кладет очередную плитку вертикально – длинной стороной с севера на юг, так что ее северная половина совпадает с клеткой, на которой робот стоит в данный момент.

Получив же команду “S”, робот не кладет плитку.

После обработки команды робот передвигается на одну клетку на юг. Если он выходит за пределы дорожки, то он сдвигается на одну клетку на

восток, и едет в самую северную клетку нового столбца.

Дорожка считается правильно замощенной, если каждая ее клетка покрыта ровно одной плиткой, и ни одна плитка не выходит за границы дорожки. Программа называется правильной для конкретного m, если существует такое n, что робот, выполняющий эту программу на дорожке $m \times n$, правильно ее замостит и (выполнив последнюю команду) сойдет с юго-восточной клетки дорожки.

Например, программа "HHSS" корректна для m=2, поскольку робот, выполняющий ее на дорожке 2×2 замостит дорожку правильно и покинет юго-восточную клетку. В то же время программа "HHV" не корректна для m=2 по двум причинам: во-первых, первые две плитки пересекаются с третьей, а во-вторых, робот не покидает дорожку (ни при каком n).

Теперь главного программиста фирмы попросили написать программу, которая верифицирует правильность программ для этого робота. Но главный программист недавно прослушал лекцию по конечным автоматам, и решил составить автомат, принимающий те и только те строчки, которые являются корректными программами для конкретного m.

Однако главный программист уехал на зимнюю школу по программированию, и этот проект поручили вам. Теперь вам по данному m нужно построить конечный автомат, распознающий корректные (для этого m) программы. Поскольку проект коммерческий, автомат должен быть детерминированным.

Формат входного файла

Во входном файле содержится целое число $m (1 \leq m \leq 10)$.

Формат выходного файла

Выведите ДКА, принимающий корректные программы для данного m, в формате, описанном в задаче "A", но без первой строки (строки с алфавитом). Число состояний в построенном автомате не обязано быть минимальным, но не должно превышать 20 000.

Пример

d.in	d.out
2	5 1 1 1 2 3 4 5 4 4 4 4 1 4 4 4 4 4 3

Задача E. Непоглощающий ДКА

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Иногда для упрощения автомата вводится понятие непоглощающих ребер. Это значит, что в дополнение к функции переходов δ вводится также функция χ: $Q \times \Sigma \to \{0, 1\}$. Тогда при совершении перехода из состояния q по символу с первый символ из входной строки удаляется, только если $\chi(q,c) = 0$. Если же $\chi(q,c) = 1$, то входная строка остается без изменений, и следующий переход производится из нового состояния, но по тому же символу c. В первом случае говорят, что произошел переход по поглощающему ребру, во втором – по непоглощающему.

По определению такой автомат допускает строку α, если после некоторого числа шагов он попадает в терминальное состояние, а строка оказывается пустой.

Дан ДКА с непоглощающими ребрами. Сколько строк длины n он допускает?

Формат входного файла

В начале входного файла содержится описание пятерки$\langle Q,\ \Sigma,\ \delta,\ q_0,\ F\rangle$ автомата A в формате, описанном в задаче "A". Затем следуют $|Q|$ строк, содержащие по $|\Sigma|$ чисел, – описание функции χ. Последняя строчка содержит целое число n $(1 \le n \le 10)$. В данной задаче $|Q| \le 1000$.

Формат выходного файла

Выведите одно число – количество строк длины n над алфавитом Σ, которые принимаются заданным автоматом.

Пример

e.in	e.out
ab 2 1 1 2 2 1 1 2 0 1 0 0 3	2

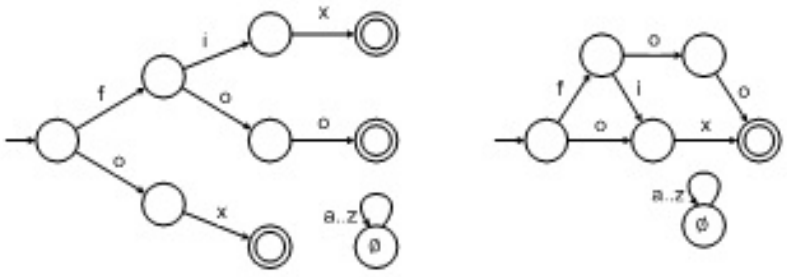
Данный автомат принимает две строки длины 3 - “aaa” и “abb”.

Задача F. Распознавание конечного языка

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Зафиксируем некоторый язык L, состоящий из конечного числа слов.

Легко построить ДКА, принимающий этот язык. Например, структура данных “бор” расширяется до искомого ДКА тривиальным образом (см. рис. слева; все не изображенные на иллюстрации переходы ведут в состояние $\emptyset$). Однако всегда существует минимальный ДКА, принимающий данный язык (см. рис. справа; все не изображенные на иллюстрации переходы ведут в состояние $\emptyset$).



Найдите число состояний в минимальном ДКА, принимающем данный конечный язык.

Формат входного файла

В первой строке входного файла содержится целое число n — количество слов в языке L ($1 \le n \le 50$).

В следующих n строках содержатся слова языка L, каждое из которых состоит из строчных букв латинского алфавита и имеет длину от 1 до 30.

Все слова во входном файле различны.

Формат выходного файла

Выведите одно число — минимальное возможное число состояний в ДКА, принимающем язык, приведенный во входном файле.

Примеры

f.in	f.out
3 fix foo ox	6
4 a ab ac ad	4

Задача G. Регулярное выражение

Имя входного файла:	g.in
Имя выходного файла:	g.out
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Everybody start back. I know regular expressions.

$http://xkcd.com/208$

Дано регулярное выражение R над алфавитом “a”, “b”.
Постройте минимальный ДКА, принимающий тот же язык, что и R.

Формат входного файла

Входной файл содержит регулярное выражение R. В записи регулярного выражения могут использоваться следующие конструкции:

- “a”, “b” - задает язык, состоящий из одного слова - соответствующей буквы;
- “0” (цифра “ноль”) – задает язык $\emptyset$ (пустое множество);
- “e” – задает язык $\{\varepsilon\}$, состоящий из одного слова – пустого;
- *(x)* – задает тот же язык, что и регулярное выражение x;
- x^* – задает замыкание Клини языка, задаваемого регулярным выражением x;

- xy – конкатенация языков, задаваемых регулярными выражениями. x и y;
- $x|y$ – задает язык, состоящий из объединения языков, задаваемых x и y.

Операции перечислены в порядке уменьшения приоритета.
Длина регулярного выражения R не превышает 10 символов.

Формат выходного файла

Выведите описание минимального ДКА, эквивалентного регулярному выражению R, в формате, описанном в задаче "A", но без первой строки (строки с алфавитом).

Примеры

g.in	g.out
(a|b)*(aa|bb)(a|b)*	4 4 1 3 3 2 1 3 3 3 1 2
(a|e)(ba)*(b|e)	4 4 3 1 2 4 3 2 1 3 3 3 1 2
ab0ab	1 1 0 1 1

Первое регулярное выражение задает язык, состоящий из строк, в которых встречаются две одинаковые буквы подряд. Второе выражение — язык из строк, в которых **не** встречаются две одинаковые буквы подряд. Третье регулярное выражение задает пустой язык.

Задача H. Автомат-собеседник для Эллочки-людоедки

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

> Словарь Вильяма Шекспира, по подсчету исследова телей, составляет 12 000 слов. Словарь негра из людо едского племени “МумбоЮмбо” составляет 300 слов. Эллочка Щукина легко и свободно обходилась тридца тью. *Илья Ильф* и *Евгений Петров*, *“12 стульев”*. Хо-хо! (Выражает, в зависимости от обстоятельств, иро нию, удивление, восторг, ненависть, радость, презрение и удовлетворенность.)
>
> *Илья Ильф и Евгений Петров, “12 стульев”*.

Приведите ДКА над алфавитом “x”, “o”, принимающий ровно n различных слов, среди которых никакое слово не является префиксом никакого другого слова (такое множество слов называется префиксный код). ДКА должен содержать минимальное возможное число состояний (среди всех ДКА, принимающих префиксные коды размера n).

Формат входного файла

Входной файл содержит целое число n $(1 \le n \le 40)$.

Формат выходного файла

Выведите описание искомого автомата в формате, описанном в задаче “A”, но без первой строки (строки с алфавитом).

Примеры

h.in	h.out
30	8 1 1 7 2 2 3 5 4 4 5 5 6 7 7 7 8 8 8 8

Данный автомат принимает язык (x|o)(x(x|o)(x|o)|o)(x(x|o)|o), в котором ровно 30 слов. В частности, слово “хохо” принимается данным автоматом.

> Ого! (Ирония, удивление, восторг, ненависть, радость, пре зрение и удовлетворенность.)
>
> *Илья Ильф и Евгений Петров*, *“12 стульев”*.

Задача I. Минимизация ДКА

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	10 с
Ограничение по памяти:	256 Мб

Дан детерминированный конечный автомат A.

Постройте детерминированный конечный автомат, принимающий тот же язык, что и A, и имеющий наименьшее возможное число состояний.

Формат входного файла

Первая строка входного файла содержит алфавит Σ, который является непустым подмножеством латинского алфавита (все буквы строчные).

Следующая строка содержит число $|Q|$ - количество состояний автомата ($1 \leq |Q| \leq 50$). Состояния нумеруются числами от 1 до $|Q|$. Следующая строка содержит число q_0 ($1 \leq q_0 \leq Q$) — номер начального состояния, затем число $|F|$ — количество терминальных состояний, затем $|F|$ чисел от 1 до

$|Q|$ — номера терминальных состояний. Следующие $|Q|$ строк содержат по $|\Sigma|$ чисел — описание функции δ.

Формат выходного файла

Выведите описание искомого детерминированного конечного автомата в формате, описанном выше, но без первой строки (строки с алфавитом).

Пример

i.in	i.out
ab 5 1 2 2 3 2 3 1 4 4 1 3 2 5 5	2 1 1 2 2 2 1 1

Данный автомат принимает слова нечетной длины из букв "a" и "b".

Разбор задачи I. Минимизация ДКА

Процесс минимизации ДКА заключается в поиске всех пар различимых состояний.

Два состояния называются q_1 и q_2 различимыми, если существует слово s, такое что при чтении s из q_1 автомат переходит в терминальное состояние, а из q_2 – в нетерминальное, или наоборот. Говорят, что слово s различает эти два состояния. Эквивалентное определение такое: состояния различимы, если у них различны правые контексты.

Ясно, что терминальное и нетерминальное состояние различимы – их различает пустое слово. (Альтернативно: пустое слово входит в правый контекст терминального состояния, но не входит в правый контекст нетерминального).

Кроме того, пусть по некоторому символу c из состояния q_1 ведет ребро в q_3, а из q_2 – в q_4. И пусть q_3 и q_4 различимы, например, их различает слово s. Тогда состояния q_1 и q_2 различимы: их различает слово cs.

Рассмотрим граф, в котором вершины — это пары состояний автомата. А ребра соответствуют **обратным** переходам по конкретному символу. Например, в предыдущем абзаце описано ребро из вершины (q_3, q_4) в вершину (q_1, q_2) (по символу c, что уже не имеет значения). Тогда в этом графе:

1. Вершины (терминальное состояние, нетерминальное состояние) и (нетерминальное состояние, терминальное состояние) это различимые пары.

2. Из вершины, соответствующей различимой паре, ведут ребра в вершины, также соответствующие различимым парам.

Следовательно, чтобы узнать все пары различимых состояний, следует обойти данный граф, начиная из вершин, описанных в п. 1; требуется найти все вершины, достижимые из них. Для этого подходит любой алгоритм обхода в графа, например, обход в глубину. Теперь, зная все пары различимых состояний, можно минимизировать автомат. Для этого отметим, что отношение "два состояния неразличимы" является отношением эквивалентности (это несложно формально доказать). В частности, если p и q неразличимы, и q и r неразличимы, то p и r неразличимы.

Соответственно, можно выделить классы эквивалентности по этому отношению. Все неразличимые состояния "схлопываются" в один класс эквивалентности — этот класс будет соответствовать **одному** состоянию искомого минимального ДКА.

Переходы между новыми состояниями строятся несложно: если в изначальном ДКА из p в q был переход по символу c, то в новом автомате должен быть переход по символу c из состояния, соответствующего классу эквивалентности, содержащему p, в аналогичный для q. Начальные и терминальные состояния определяются в новом автомате аналогично — это классы эквивалентности, содержащие соответственно начальные и терминальные состояние изначального ДКА.

Следует также отметить, что при минимизации обязательно отсечь все состояния, недостижимые из начального. Это можно сделать как в оригинальном ДКА, так и в построенном минимальном.

Время работы алгоритма составляет $O(|Q|^2)$.

Задача J. Обращение ДКА

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Дан детерминированный конечный автомат A. Пусть L – язык, который он принимает. Постройте детерминированный конечный автомат, принимающий язык LR – язык, состоящий из слов языка L, прочитанных справа налево.

Формат входного файла

Входной файл содержит описание автомата A в формате, описанном в задаче "A". В данной задаче $|Q| \leq 10$.

Формат выходного файла

Выведите описание искомого детерминированного конечного автомата в аналогичном формате, но без первой строки (строки с алфавитом). Число состояний в построенном автомате не обязано быть минимальным, но не должно превышать 20 000.

Пример

j.in	j.out
ab 5 1 1 4 2 2 3 3 4 5 4 4 5 5	8 1 4 2 4 6 8 5 1 5 1 6 2 6 2 7 3 7 3 8 4 8 4 2

Данный во входном файле автомат принимает слова из букв "a" и "b", у которых третий символ – "a". В выходной файл следует вывести описание автомата, принимающего обращенный язык: множество слов, у которых третий **с конца** символ – "a".

Разбор задачи J. Обращение ДКА

Напрашивающийся тривиальный подход: обратить направления ребер в данном ДКА, а также назвать начальное состояние терминальным, а терминальные – начальными. . .

Стоп. Начальное состояние (как в ДКА, так и в НКА и в ε-НКА) может быть только одно. Это первая проблема, вторая - из одного состояния по одному символу может исходить несколько ребер. Вторая проблема решается бюрократическим путем: объявим полученный автомат недетерминированным, тогда наличие нескольких исходящих ребер корректно. Первая проблема решается следующим образом: заведем дополнительное новое начальное состояния, из которого проведем ε-ребра в те состояния, которые

хотелось бы назвать начальными. Итак, получился полноценный (делающий то, что требуется) ε-НКА.

Осталось получить эквивалентный ему ДКА. Для этого рассмотрим следующее множество состояний: каждое состояние искомого ДКА – это подмножество состояний ε-НКА.

Начальным состоянием ДКА назовем ε-замыкание начального состояние ε-НКА. Терминальными состояниями – те множества, которые содержат хотя бы одно терминальное состояние ε-НКА.

Наконец, чтобы совершить переход по символу c из данного множества, следует совершить переходы из всех состояний ε-НКА – элементов этого множества, рассмотреть их ε-замыкания и взять их объединение.

Полученный ДКА может иметь экспоненциально больше состояний, чем оригинальный. Так, например, автомат, принимающий слова, у которых n-я буква – “a” имеет $O(n)$ состояний, а автомат, принимающий слова, у которых n-я с конца буква - “a” имеет $O(2^n)$ состояний. Действительно, неформально говоря, он обязан в каждый момент времени “помнить” последние n букв прочитанной строки – каждая из них, когда окажется на n-й с конца позиции будет принципиально важно.

Время работы алгоритма составляет $O(|Q|^2 \cdot 2^{|Q|})$.

Задача K. ДКА, проверяющий делимость

Имя входного файла:	`k.in`
Имя выходного файла:	`k.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

В условиях данной задачи разрешим в двоичной записи чисел присутствие ведущих нулей. Кроме того разрешим записывать число 0 в двоичной записи как ε (как пустую строку).

Таким образом, строки “101” и “000101” являются двоичными записями числа 5, а строки “000000” и “” – двоичными записями числа 0. Дано три различных положительных целых чисел a, b и c.

Постройте ДКА над алфавитом “0”, “1”, принимающий те и только те строки, которые являются двоичными записями чисел, делящихся на хотя бы одно число из $\{a, b, c\}$.

Число состояний в построенном автомате не обязано быть минимальным, но не должно превышать 20 000. Гарантируется, что существует искомый ДКА с не более чем 1 000 состояний.

Формат входного файла

В первой строке входного файла содержится число |S| ($1 \leq |S| < 1000$) - мощность множества S. Затем следуют |S| различных положительных нечетных чисел, не превышающих 10^9

Формат выходного файла

Выведите описание искомого автомата в формате, описанном в задаче "I", но без первой строки (строки с алфавитом).

Пример

k.in	k.out
3 3 5 15	15 15 7 3 5 6 9 10 12 15 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1

Разбор задачи K. ДКА, проверяющий делимость

Пусть в данном множестве содержатся два числа a и b, такие что a делится на b. Тогда a можно исключить из рассмотрения – все числа, делящиеся на a, делятся и на b тоже, так что достаточно проверять делимость на b. Итак, "жадно" удалим все числа, делители которых также есть в множестве.

Теперь все оставшиеся числа важны – прочитав строку s, мы обязаны знать остаток от деления уже прочитанного числа в двоичной записи на все эти числа. Или, что эквивалентно, знать остаток от деления прочитанного числа на НОК этих чисел (обозначим его m).

Таким образом, состояние искомого ДКА соответствует остатку от деления уже прочитанного двоичного числа на m. Начальное состояние – остаток 0. Терминальные – остатки, которые делятся на хотя бы одно число из множества.

Переход также несложен: пусть до чтения символа c остаток равнялся r. Тогда после считывания c (c = 0 или 1), новый остаток равняется $(2r + c) \bmod m$.

Задача L. Укладка плиток

Имя входного файла:	`l.in`
Имя выходного файла:	`l.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Фирма “Автоматическая смена пола” разработала робота, который замощает плиткой дорожки. У робота есть неограниченный запас плиток размером 2×1. Дорожка, которую надо замостить имеет размеры $m \times n$, то есть состоит из $m \cdot n$ клеток. Здесь m – ширина дорожки, n – ее длина. Дорожка идет с запада на восток.

Программа для робота – это последовательность команд “H”, “V” и “S”. Робот начинает выполнять ее, стоя на северо-западной клетке дорожки. Получив команду “H”, робот кладет очередную плитку горизонтально – длинной стороной с запада на восток, так что ее западная половина совпадает с клеткой, на которой робот стоит в данный момент.

Получив команду “V”, робот кладет очередную плитку вертикально – длинной стороной с севера на юг, так что ее северная половина совпадает с клеткой, на которой робот стоит в данный момент.

Получив же команду “S”, робот не кладет плитку.

После обработки команды робот передвигается на одну клетку на юг. Если он выходит за пределы дорожки, то он сдвигается на одну клетку на восток, и едет в самую северную клетку нового столбца.

Дорожка считается правильно замощенной, если каждая ее клетка покрыта ровно одной плиткой, и ни одна плитка не выходит за границы дорожки. Программа называется правильной для конкретного m, если существует такое n, что робот, выполняющий эту программу на дорожке $m \times n$, правильно ее замостит и (выполнив последнюю команду) сойдет с юго-восточной клетки дорожки.

Например, программа “HHSS” корректна для m=2, поскольку робот, выполняющий ее на дорожке 2×2 замостит дорожку правильно и покинет юго-восточную клетку. В то же время программа “HHV” не корретна для m=2 по двум причинам: во-первых, первые две плитки пересекаются с третьей, а во-вторых, робот не покидает дорожку (ни при каком n).

Теперь главного программиста фирмы попросили написать программу, которая верифицирует правильность программ для этого робота. Но глав-

ный программист недавно прослушал лекцию по конечным автоматам, и решил составить автомат, принимающий те и только те строчки, которые являются корректными программами для конкретного m.

Однако главный программист уехал на зимнюю школу по программированию, и этот проект поручили вам. Теперь вам по данному m нужно построить конечный автомат, распознающий корректные (для этого m) программы. Поскольку проект коммерческий, автомат должен быть детерминированным.

Формат входного файла

Во входном файле содержится целое число $m (1 \leq m \leq 10)$.

Формат выходного файла

Выведите ДКА, принимающий корректные программы для данного m, в формате, описанном в задаче "I", но без первой строки (строки с алфавитом). Число состояний в построенном автомате не обязано быть минимальным, но не должно превышать 20 000.

Пример

l.in	l.out
2	5 1 1 1 2 3 4 5 4 4 4 4 1 4 4 4 4 4 3

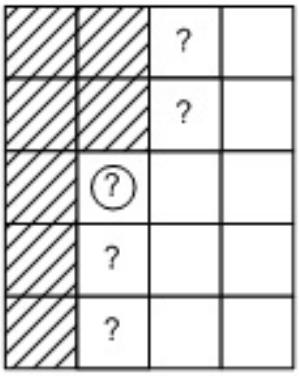
Разбор задачи L. Укладка плиток

Задача относится (в некотором смысле) к теме "динамическое программирование по профилю".

Рассмотрим робота, корректно замощающего некоторую дорожку $m \times n$. Как выглядит не до конца замощенная дорожка в некоторый момент времени? Как следует из документации, приведенной в условии задачи, робот никогда не покрывает плитками клетки, на которых он уже был: если он кладет плитку, то она занимает клетку, на которой он стоит сейчас, и клетку, на которой ему предстоит побывать в будущем.

Следовательно, (если программа корректна) когда робот находятся в некоторой клетке, все ранее посещенные клетки должны быть уже замощены. Если это не так, то программа точно не корректна – ДКА следует перевести в “дьявольское состояние $\emptyset$”>.

Таким образом, все ранее посещенные клетки замощены. Рассмотрим теперь клетки, которые робот посетит в дальнейшем, а так же ту, на которой он стоит сейчас. Из этих клеток m теоретически могут оказаться замощенными.



Робот обозначен кружком, точно замощенные клетки заштрихованы,возможно замощенные – обозначены вопросом.

Эту информацию (какие из этих клеток закрашены) необходимо знать. Это m бит. Кроме того, необходимо знать координату “излома”. Именно эта информация и описывает состояния искомого ДКА. Начальное состояние – отсутствие “излома” + все клетки, отмеченные вопросами, незамощены. Терминальное состояние, что интересно, одно – оно же.

Наконец, чтобы узнать переход из данного состояния по данной команде, следует выполнить эту команду и закодировать полученное состояние. Здесь надо аккуратно рассмотреть все возможные некорректные действия робота – в каждом из таких случаев ДКА должен переходить в дьявольское состояние.

Следует отметить, что работу с состояниями можно реализовать весьма эффективно, если использовать битовые маски. Это частый прием при работе с “профилями”, такими как в этой задаче.

Время работы алгоритма и размер получаемого автомата – $O(n \cdot 2^n)$.

Задача M. Непоглощающий ДКА

Имя входного файла:	`m.in`
Имя выходного файла:	`m.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Иногда для упрощения автомата вводится понятие непоглощающих ребер. Это значит, что в дополнение к функции переходов δ вводится также функция χ: $Q \times \Sigma \to \{0, 1\}$. Тогда при совершении перехода из состояния q по символу с первый символ из входной строки удаляется, только если $\chi(q,c) = 0$. Если же $\chi(q,c) = 1$, то входная строка остается без изменений, и следующий переход производится из нового состояния, но по тому же символу c. В первом случае говорят, что произошел переход по поглощающему ребру, во втором – по непоглощающему.

По определению такой автомат допускает строку α, если после некоторого числа шагов он попадает в терминальное состояние, а строка оказывается пустой.

Дан ДКА с непоглощающими ребрами. Сколько строк длины n он допускает?

Формат входного файла

В начале входного файла содержится описание пятерки $\langle Q, \Sigma, \delta, q_0, F\rangle$ автомата A в формате, описанном в задаче “A”>. Затем следуют $|Q|$ строк, содержащие по $|\Sigma|$ чисел, – описание функции χ. Последняя строчка содержит целое число n $(1 \le n \le 60)$. В данной задаче $|Q| \le 1000$.

Формат выходного файла

Выведите одно число – количество строк длины n над алфавитом Σ, которые принимаются заданным автоматом.

Пример

m.in	m.out
ab 2 1 1 2 2 1 1 2 0 1 0 0 3	2

Данный автомат принимает две строки длины 3 - “aaa” и “abb”.

Разбор задачи M. Непоглощающий ДКА

Сперва надлежит узнать, в какое состояние в итоге перейдет автомат, который в состоянии q получил на вход символ c – с учетом непоглощающих ребер. Это — несложная задача на графе, решаемая обходом в глубину. В результате мы для каждой пары (q, c) либо $\delta(q, c)$ узнаем состояние, в которое автомат приходит, удалив, наконец, символ, либо узнаем, что автомат “зависает” — оказывается в бесконечном непоглощающем цикле. Во второй ситуации можно сказать, что автомат просто переходит в дьявольское состояние $\emptyset$.

Так мы получили ДКА без непоглощающих ребер, эквивалентный исходному. Задача — посчитать количество слов длины n, которые он принимает. Задача решается методом динамического программирования.

Пусть A_{qi} - число слов длины i, прочитав которые автомат приходит в состояние q. (Другими словами: число слова длины i в левом контексте состояния q).

Со словами длины 0 все просто: есть одно такое слово, после его прочтения автомат оказывается в начальном состоянии.

Рассмотрим некоторое A_{qi}. Пусть автомат считал уже какое-то слово из i букв и находится сейчас в состоянии q (он мог сделать это ровно A_{qi} различными способами). Теперь есть $|\Sigma|$ вариантов дальнейшего действия — на вход может прийти любая буква. Таким образом, если есть переход в состояние r, то к числу $A_{r,i+1}$ следует прибавить A_{qi}. Если переходов несколько, то прибавить следует несколько раз.

Наконец, число слов длины n, принимаемых автоматом — это сумма чисел A_{qi} по всем терминальным состояниям q.

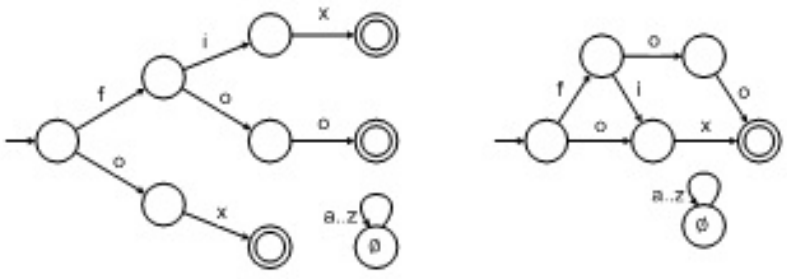
При данных ограничениях это число не влезает в 64-битный целочисленный тип данных, требуется реализовывать собственную “длинную арифметику” или пользоваться существующей библиотекой.

Задача N. Распознавание конечного языка

Имя входного файла:	`n.in`
Имя выходного файла:	`n.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Зафиксируем некоторый язык L, состоящий из конечного числа слов.

Легко построить ДКА, принимающий этот язык. Например, структура данных “бор” расширяется до искомого ДКА тривиальным образом (см. рис. слева; все не изображенные на иллюстрации переходы ведут в состояние $\emptyset$). Однако всегда существует минимальный ДКА, принимающий данный язык (см. рис. справа; все не изображенные на иллюстрации переходы ведут в состояние $\emptyset$).



Найдите число состояний в минимальном ДКА, принимающем данный конечный язык.

Формат входного файла

В первой строке входного файла содержится целое число n — количество слов в языке L ($1 \le n \le 50$).

В следующих n строках содержатся слова языка L, каждое из которых состоит из строчных букв латинского алфавита и имеет длину от 1 до 30.

Все слова во входном файле различны.

Формат выходного файла

Выведите одно число — минимальное возможное число состояний в ДКА, принимающем язык, приведенный во входном файле.

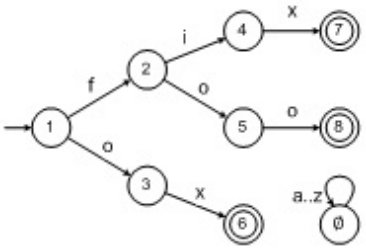
Примеры

n.in	n.out
3 fix foo ox	6
4 a ab ac ad	4

Разбор задачи N. Распознавание конечного языка

Вспомним алгоритм минимизации ДКА. Два состояния различимы, если у них разные правые контексты.

Соответственно, автомат, тривиально получаемый из бора, может не быть минимальным, если в нем есть состояния с одинаковыми правыми контекстами. Их следует найти и объединить.



Но что такое правый контекст в автомате-боре? Это множество слов, которые читаются из данной вершины. Или, другими словами, бор, начинающийся из данной вершины.

Например, на иллюстрации правый контекст вершины 3 – язык “x”. И вершины 4 – он же. Нагляднее всего это объяснять тем, что поддеревья этих двух вершин совпадают. Значит, поставленная задача – объединить все одинаковые поддеревья.

Будем присваивать деревьям идентификаторы (номера), причем одинаковым деревьям – одинаковые, разным – разные.

Начнем с пустого дерева, ему присвоим идентификатор 0. Поймем, как узнать класс эквивалентности (идентификатор) произвольного дерева. Он однозначно задается 26 числами – идентификатором поддерева по ребру “a” (если такого ребра нет, то поддерева не существует, назовем это идентификатором -1), по ребру “b” и так далее до “z”.

Будем хранить все таки 26-ки чисел в некоторой коллекции, например в хэш-таблице. И для каждой 26-ки будем хранить ее идентификатор. Тогда решение задачи заключается в том, чтобы для всех вершин бора найти идентификаторы поддеревьев, растущих из них. Количество разных идентификаторов задает число состояний в минимальном ДКА. Плюс одно состояние – дьявольское состояние $\emptyset$.

При обработки каждой вершины, все ее поддеревья должны быть уже обработаны. Этого можно добиться а) запуская соответствующую процедуру рекурсивно; б) обрабатывая вершины в порядке уменьшения расстояния от них до корня. Время работы алгоритма пропорционально размеру бора (или размеру входного файла).

Задача O. Регулярное выражение

Имя входного файла:	`o.in`
Имя выходного файла:	`o.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Everybody start back. I know regular expressions.

$http://xkcd.com/208$

Дано регулярное выражение R над алфавитом “a”, “b”.
Постройте минимальный ДКА, принимающий тот же язык, что и R.

Формат входного файла

Входной файл содержит регулярное выражение R. В записи регулярного выражения могут использоваться следующие конструкции:

- “a”, “b” - задает язык, состоящий из одного слова - соответствующей буквы;
- “0” (цифра “ноль”) – задает язык $\emptyset$ (пустое множество);
- “e” – задает язык $\{\varepsilon\}$, состоящий из одного слова – пустого;
- *(x)* – задает тот же язык, что и регулярное выражение x;
- x^* – задает замыкание Клини языка, задаваемого регулярным выражением x;

- xy – конкатенация языков, задаваемых регулярными выражениями. x и y;
- $x|y$ – задает язык, состоящий из объединения языков, задаваемых x и y.

Операции перечислены в порядке уменьшения приоритета.
Длина регулярного выражения R не превышает 20 символов.

Формат выходного файла

Выведите описание минимального ДКА, эквивалентного регулярному выражению R, в формате, описанном в задаче "I", но без первой строки (строки с алфавитом).

Примеры

o.in	o.out
(a|b)*(aa|bb)(a|b)*	4 4 1 3 3 2 1 3 3 3 1 2
(a|e)(ba)*(b|e)	4 4 3 1 2 4 3 2 1 3 3 3 1 2
ab0ab	1 1 0 1 1

Первое регулярное выражение задает язык, состоящий из строк, в которых встречаются две одинаковые буквы подряд. Второе выражение — язык из строк, в которых **не** встречаются две одинаковые буквы подряд. Третье регулярное выражение задает пустой язык.

Разбор задачи O. Регулярное выражение

Для любого регулярного выражения R существует эквивалентный ему ε-НКА, в котором:

- Ровно одно терминального состояние;
- Нет ребер, исходящих из терминального состояния;
- Нет ребер, ведущих в начальное состояние.

Такой автомат строится индуктивно. Его легко построить для регулярных выражений “a”, “b”, “0”, “e”, а для замыкания Клини, конкатенации и объединения существуют несложные конструкции, позволяющие построить искомый ε-НКА, используя ε-НКА для частей данного регулярного выражения. Этот алгоритм подробно описан в книге Хоп-крофта, Мотвани и Ульмана “Введение в теорию автоматов”.

В данной задаче требуется разобрать регулярное выражение и применить данный алгоритм. Поскольку регулярное выражение небольшое, разборщик можно реализовывать и не самый эффективный. Построение ε-НКА также занимает мало времени. Основная сложность — преобразовать его в минимальный ДКА. Описание этого процесса приведено в разборе задачи “B”.

Алгоритм получается экспоненциальным, однако следует активно использовать тот факт, что в ДКА экспоненциального размера далеко не все состояния достижимы из начального, в силу специфики задачи.

Задача P. Автомат-собеседник для Эллочки-людоедки

Имя входного файла:	`p.in`
Имя выходного файла:	`p.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

> Словарь Вильяма Шекспира, по подсчету исследова
> телей, составляет 12 000 слов. Словарь негра из людо
> едского племени “МумбоЮмбо” составляет 300 слов.
> Эллочка Щукина легко и свободно обходилась тридца
> тью. *Илья Ильф* и *Евгений Петров*, *“12 стульев”*.
> Хо-хо! (Выражает, в зависимости от обстоятельств, иро
> нию, удивление, восторг, ненависть, радость, презрение
> и удовлетворенность.)
>
> *Илья Ильф и Евгений Петров, “12 стульев”*.

Приведите ДКА над алфавитом “x”, “o”, принимающий ровно n различных слов, среди которых никакое слово не является префиксом никакого другого слова (такое множество слов называется префиксный код). ДКА

должен содержать минимальное возможное число состояний (среди всех ДКА, принимающих префиксные коды размера n).

Формат входного файла

Входной файл содержит целое число n $(1 \leq n \leq 200)$.

Формат выходного файла

Выведите описание искомого автомата в формате, описанном в задаче "I", но без первой строки (строки с алфавитом).

Примеры

p.in	p.out
30	8 1 1 7 2 2 3 5 4 4 5 5 6 7 7 7 8 8 8 8

Данный автомат принимает язык (x|o)(x(x|o)(x|o)|o)(x(x|o)|o), в котором ровно 30 слов. В частности, слово "хохо" принимается данным автоматом.

> Ого! (Ирония, удивление, восторг, ненависть, радость, пре зрение и удовлетворенность.)
>
> *Илья Ильф и Евгений Петров*, *"12 стульев"*.

Разбор задачи P. Автомат-собеседник для Эллочки-людоедки

Утверждение. Пусть A — минимизированный ДКА, принимающий конечный язык. Этот язык префиксный тогда и только тогда, когда в A ровно одно терминальное состояние.

Доказательство. Пусть язык префиксный, но в A хотя бы два терминальных состояния.

Первый случай. Нет терминального состояния из которого достижимо (другое или то же самое) терминальное состояние. Тогда правый контекст каждого терминального состояния — ε. Значит, все терминальные состояния эквивалентны, автомат не минимизированный, противоречие.

Второй случай. Из терминального состояния p достижимо терминальное состояние q. Так как автомат минимизированный, то p достижимо из начальной вершины (иначе можно его удалить). Пусть оно достижимо по слову s. Пусть q достижимо из p по слову t. Тогда автомат принимает и слово s, и слово st. Следовательно, язык автомата не префиксный — противоречие.

В другую сторону. Пусть в A одно терминальное состояние q. Допустим, что язык не префиксный. Рассмотрим слова языка s и st. Автомат, прочитав слово s, принимает его, значит, оказывается в q. Но и прочитав слово st, он также оказывается в q. Значит, слово t переводит автомат из состояния q в него же. Но тогда автомат принимает слова stt, $sttt$ и так далее. Язык не конечен. Противоречие. **Конец доказательства.**

Итак, искомый автомат имеет одно конечное состояние. Рассмотрим правый контекст этого состояния. Это язык ε, мощность этого языка – 1.

Отсортируем состояния автомата (кроме дьявольского) топологически (легко убедиться, что среди них нет цикла). Рассмотрим некоторое состояние r, которое ведет по одному символу в p, а по другому – в q. Правый контекст r складывается из частей, и его мощность равна сумме мощности правого контекста p и правого контекста q. Если одно или два из этих состояний – дьявольские, то мощность соответствующего правого контекста — 0, этот случай не является исключением.

Будем двигаться в соответствии с топологической сортировкой от конечного состояния с начальному и следить за мощностью правых контекстов. Каждая очередная мощность — сумма каких-то двух уже имеющихся чисел. А в итоге надо получить n.

Это задача — классическая, она называется задача об “оптимальный алгоритме возведения в степень”. Действительно, например вычислить a^30 можно следующим образом:

- $a \cdot a = a^2$;
- $a^2 \cdot a = a^3$;

- $a^3 \cdot a^3 = a^6$;
- $a^6 \cdot a^6 = a^{12}$;
- $a^{12} \cdot a^3 = a^{15}$;
- $a^{15} \cdot a^{15} = a^{30}$.

Эти семь промежуточных значений в точности соответствуют состояниям автомата, приведенного в условии (кроме дьявольского состояния). Минимизация числа промежуточных действий при возведении в степень — сложная задача; для степеней не более 200 ее можно за разумное время решить, применив перебор с отсечениями.

День третий (20.02.2009 г.)
Контест Виталия Гольдштейна

Об авторе...

Студент Саратовского Государственного Университета. Чемпион России на олимпиаде по информатике среди школьников 2004. Золотая медаль на международной олимпиаде школьников по информатике 2004 (IOI Athens 2004). 17 место чемпионата мира по программированию среди студентов (ACM ICPC Shanghai 2005). Серебряная медаль чемпионата мира по программированию среди студентов (ACM ICPC Tokyo 2007). Участник финалов (1 из 48 со всего мира) TopCoder Open (дважды), TopCoder Collegiate Open, Global Google Code Jam, Google Code Jam Europe (дважды). Член научного комитета всероссийской олимпиады школьников по информатике и сборов по подготовке к международной олимпиаде школьников. Преподаватель летней компьютерной школы. Стажер компании Google Inc..

Теоретический материал. Потоки в сетях

1. Базовые определения

Сеть - ориентированный граф $G(V, E)$, в котором каждому ребру $(u, v) \in E$ преписана пропускная способность $c(u, v)$. Если $(u, v) \notin E$, то $c(u, v) = 0$. Выделяются две вершины: источник s и сток t. Поток - $f : V \times V \to R$ со следующими свойствами для вершин u и v:

- "Ограничение пропускной способности": $f(u, v) \leq c(u, v)$. Поток не может превысить пропускную способность.
- "Антисимметричность": $f(u, v) = -f(v, u)$. Поток из u в v должен быть противоположным потоку из v в u.
- "Сохранение потока": $\sum_{w \in V - \{s,t\}} f(u, w) = 0$.

"Остаточная пропускная способность" ребра $c_f(u,v) = c(u,v) - f(u,v)$. "Остаточная сеть" обозначается $G_f(V, E_f)$. В остаточной сети может быть ребро из u в v, даже если его нет в исходной сети. "Увеличивающий путь" - это путь (u_1, u_2, K, u_k) в остаточной сети, где $u_1 = s$, $u_k = t$, и $c_f(u_i, u_{i+1}) > 0$.

Величиной потока назовем: $\sum_{w \in V} f(s,w)$. Разрезом назовем такое разбиение множества V на два множества S и T, что выполнены условия:

- $S \cap T = \varnothing$,
- $S \cup T = V$,
- $s \in S$,
- $t \in T$.

Пропускная способность разреза $C(S,T) = \sum_{u \in S, v \in T} c(u,v)$. Потоком через разрез $F(S,T) = \sum_{u \in S, v \in T} f(u,v)$.

Максимальный поток - поток, имеющий максимальную величину. Минимальный разрез - разрез с минимальной пропускной способностью.

2. Базовые теоремы

- Поток через любой разрез равен величине потока.
- Величина минимального разреза равна величине потока.
- *Теорема Форда-Фалкерсона*. Поток максимален тогда и только тогда, когда не существует увеличивающего пути в остаточной сети.

3. Метод Форда-Фалкерсона

Метод Форда-Фалкерсона основан на теореме Форда-Фалкерсона. Изначально установим поток пустым, то есть $f(u,v) = 0$. Найдем увеличивающий путь (u_1, u_2, K, u_k) в остаточной сети, затем найдем $t = \min_{i \in [1,k-1]} \tilde{n}_f(u_i, u_{i+1})$. После этого "пропустим поток" величиной вдоль найденного пути, то есть совершим следующие операции для всех $i \in [1, k-1]$, $f(u_i, u_{i+1}) += t$ и $f(u_{i+1}, u_i) -= t$. После этой операции величина потока увеличится на t и все свойства потока останутся выполнены. Алгоритм останавливается, когда в остаточной сети не существует дополняющего пути. Существуют примеры, что этот метод бесконечно стремится к максимальному потоку, но никогда его не достигает. Это возможно при иррациональных пропускных способностях.

При целых пропускных способностях время работы алгоритма можно оценить как $O(maxFlow^*|E|)$.

Алгоритм Эдмондса-Карпа

Алгоритм Эдмондса-Карпа отличается от метода Форда-Фалкерсона тем, что в качестве дополняющего пути выбирается самый короткий путь по количеству ребер. Благодаря такому выбору время работы алгоритма можно оценить как $O(VE^2)$. Предыдущая оценка из метода Форда-Фалкерсона так же остается верна. Асимптотика $O(VE^2)$ объясняется несколькими фактами:

1. Расстояние от истока до других вершин в остаточной сети не уменьшается при ее изменении.
2. Каждое ребро может минимальным по пропускной способности в выбранном дополняющем пути может быть $O(V)$ раз.
3. Общее количество ребер равно $O(V)$, а поиск кратчайшего пути с помощью обхода в ширину осуществляется за $O(V)$.

Такое время работы, фактически, недостижимо. Можно рассчитывать, что алгоритм будет достаточно эффективен при 100-200 вершинах.

Нахождение минимального разреза

Найдем максимальный поток. Найдем множество вершин S, достижимых из истока в остаточной сети. $C(S,V-S)=F(S,V-S)$. Пропускная способность разреза равна максимальному потоку, то есть выбранный разрез является минимальным.

Алгоритм масштабирования

Пропускной способностью дополняющего пути назовем минимальное значение пропускной способности среди ребер этого пути, т.е. величину t. В алгоритме масштабирования будем искать не произвольный дополняющий путь, а путь с пропускной способностью не менее A. После того, как не будет существовать такого дополняющего пути, продолжим поиск для $A/2$. Алгоритм закончится, когда не будет существовать никакого дополняющего пути.

Если все пропускные способности целые, то алгоритм будет работать верно, так как на последней итерации при $A=1$ будет осуществляться поиск обычного дополняющего пути.

Преимуществом этого алгоритма является возможность поиска любого дополняющего пути, который можно искать с помощью обхода в глубину. Путь нужно искать в немного измененной остаточной сети, где $c_f(u,v) = c(u,v) - f(u,v) - A + EPS$, где *EPS* добавлено, чтобы иметь возможность искать дополняющие пути с пропускной способностью ровно *A*. На практике нужно просто при поиске дополняющего пути использовать условие $c_f(u,v) \geq 0$.

Кроме этого время работы этого алгоритма $O(E^2 \log C)$, где *C* - максимальная пропускная способность одного ребра. Асимптотика $O(E^2 \log C)$ объясняется следующим образом: Пусть не существует дополняющего пути пропускной способностью не менее *A*. Найдем множество *S* вершин, достижимых из истока в измененной остаточной сети. Все ребра, ведущие в недостижимые вершины в обычной остаточной сети, имеют пропускную способность меньше *A*. То есть величина наименьшего разреза не превосходит A*E. Таким образом, на следующем шаге потребуется не более $\frac{AE}{A/2} = 2E$ шагов. Каждый шаг работает за O(E).

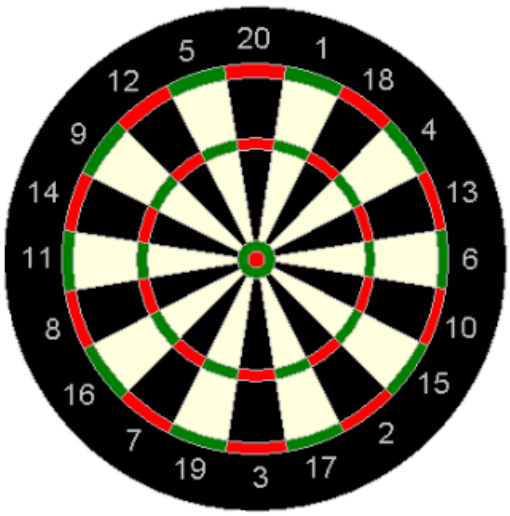
Таким образом, одна итерация работает за $O(E^2)$, всего итераций $\log C$. Итоговая асимптотика равна $O(E^2 \log C)$.

Задачи и разборы

Задача A. Дартс

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Дартс – игра, в которой игроки метают короткие стрелы (дротики) в круглую мишень, повешенную на стену. Стандартная мишень разделена на двадцать пронумерованных секторов, каждой присвоено число от 1 до 20. В центре находится “яблочко”, попадание в которое оценивается в 50 очков, окружённое зелёным кольцом вокруг него (25 очков). Внешнее узкое кольцо (кольцо “даблов”) означает удвоение числа сектора, внутреннее узкое кольцо (кольцо “треблов”) означает утроение числа секто-



ра. Количество очков, получаемое за каждый сектор, указано на рисунке. Внешнее и внутреннее узкие кольца традиционно окрашиваются в красный и зелёный цвета. Попадание дротика вне круга ограниченного внешним узким кольцом очков не приносит. На стене висит мишень, причем ее центр совпадает с точкой (0, 0). Известно, что после броска, дротик застрял в точке с координатами *(x, y)* (все координаты описываются в миллиметрах, y – обозначает координату по вертикали, x – по горизонтали). Требуется подсчитать, сколько очков получит игрок за такой бросок. Гарантируется, что после броска дротик не попал на границу между двумя различными секторами, и можно однозначно определить количество очков, набранных игроком.

Стандартные размеры мишени:

- внутренняя ширина колец “даблов” и “треблов” 8 мм;
- внутренний диаметр “яблочка” 12,7 мм;
- внутренний диаметр внешнего центрального кольца 31,8 мм;
- расстояние от центра мишени до внешней стороны кольца “даблов” 170,0 мм;
- расстояние от центра мишени до внешней стороны кольца “треблов” 107,0 мм.

Формат входного файла

Во входном файле через пробел записаны два дробных числа (с 2 знаками после запятой), описывающие координаты точки.

Формат выходного файла

Выведите единственное число — ответ на поставленную задачу.

Пример

a.in	a.out
0.00 0.00	50
0.00 102.00	60

Разбор задачи A. Дартс

В первую очередь, необходимо определить расстояние от центра мишени до данной точки, и полярный угол, считая от границы одного из секторов. Далее последовательностью операторов условия “if” определяем,

в какую из очковых зон мы попали, в случае если это яблочко, окружающее его колечко или пространство вне мишени, то сразу выводим ответ и завершаем программу. Иначе определяем, находится ли дротик в области удвоения или утроения, если это так, то коэффициент будет равен 2 или 3, иначе 1. Затем остается определить номер сектора. Всего секторов - 20, каждый из них имеет раствор ровно 18 градусов. Тогда, зная полярный угол в ранее нами введенной системе координат, можно определить в какой по порядку сектор попал наш дротик. Предварительно заполнив константный массив номерами каждого сектора, без труда можно будет получить количество очков полученных за этот сектор. Ответом будет являться количество очков сектора, умноженное на коэффициент области.

Задача B. Сочинитель стихов

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Начинающий поэт Вася разработал конечный автомат для упрощения процесса сочинения стихов. Этот автомат имеет N состояний, при этом переход из состояния в состояние осуществляется по K возможным рифмам. Автомат имеет одно начальное состояние a и одно конечное состояние b. Результатом работы сочинителя стихов является некоторая последовательность рифм, принимаемая автоматом, на которую Вася накладывает готовые стихи.

Для того, чтобы стихи не были слишком однообразными, после каждого перехода Вася несколько изменяет автомат. А именно, как только происходит переход из состояния i в состояние j по рифме k, Вася стирает все переходы, исходящие из состояния i по рифме k, а также все переходы, ведущие в состояние j по рифме k.

Требуется определить максимальный набор стихов, которые Вася может сочинить, используя свой автомат таким образом. После того, как какой-то переход стерт, он уже больше никогда в истории жизни этого автомата не будет восстановлен, так как Вася не хочет повторяться.

Формат входного файла

Первая строка входного файла содержит четыре целых числа — количество состояний автомата N ($1 \le N \le 50$), количество рифм, которые использует Вася в своих стихах K ($1 \le K \le 50$), а также номера начального и конечного состояний a и b ($1 \le a \ne b \le N$). Вторая строка содержит

одно целое число — количество переходов в автомате M ($1 \le M \le 1000$). Далее следуют M строк, каждая из которых описывает один переход в формате $u_i v_i k_i$, означающий, что из состояния u_i можно перейти в состояние v_i по рифме k_i.

Формат выходного файла

В первой строке выходного файла выведите максимальное количество стихов Z, которые Вася может сочинить с помощью своего автомата. Далее выведите Z строк, по одной для каждого стиха. Описание стиха выводите в формате

$$s_1\ k_1\ s_2\ k_2 \ldots s_{l-1}\ k_{l-1}\ s_l,$$

где $s_1 = a$, $s_l = b$, k_i — рифмы, по которым производится переход, а S_i — состояния в порядке прохода.

Пример

b.in	b.out
6 3 1 4	2
9	1 1 2 1 3 3 4
1 2 1	1 2 5 1 4
1 6 1	
2 3 1	
3 4 3	
1 5 2	
1 2 2	
5 4 1	
2 4 3	
6 4 2	

Разбор задачи B. Сочинитель стихов

Задача сводится к нахождению максимального количества путей, причем из одной вершины не выходят более одного ребра одного цвета. Сведем задачу к нахождения максимального потока. Для каждой вершин создадим *k+1* вершину в перестроенной сети. Первая вершина будет входом, остальные вершины будут представлять цвета. Из вершины входа проведем по ребру пропускной способностью 1 в каждую из k вершин, соответствующих цвету. Из вершины соответствующих цвету i проведем все ребра цвета i во входы концов ребер. Найдя максимальный поток в такой сети, получим максимальное количество путей удовлетворяющих требуемому свойству.

Задача C. Число инверсий

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Перестановка это последовательность длины n различных целых чисел от 1 до n. Например, (5, 3, 2, 1, 4) это перестановка. Инверсией перестановки p называется такая упорядоченная пара индексов *(i, j)*, что $i < j$ и $p_i > p_j$. В примере выше пара индексов (2, 4) образует инверсию, так как $2 < 4$ и $3 > 1$. Задана пара чисел n и t. Найдите количество перестановок из n элементов, в которых ровно t инверсий. Выведите наименьшую в лексикографическом порядке перестановку с t инверсиями.

Формат входного файла

В первой строке входного файла записана пара целых чисел n и t ($1 \le n \le 18$; $0 \le t \le 200$).

Формат выходного файла

В первую строку выведите количество перестановок из n элементов, которые имеют ровно t инверсий. Во вторую строку выведите наименьшую в лексикографическом порядке перестановку с заданным числом инверсий t. Если такой перестановки не существует, выведите в первую строку “0”, а во вторую – символ “-”.

Пример

c.in	c.out
3 0	1 1 2 3
3 1	2 1 3 2

Разбор задачи C. Число инверсий

Решим задачу методом динамического программирования. Научимся вычислять $D[i][j]$ — количество перестановок длины i с j инверсиями. Для этого переберём позицию наибольшего элемента (т.е. равного i). Пусть он находится в позиции k=*1*. . . i, тогда он образует ровно i-k инверсий, а остальные инверсии образуются между другими элементами перестановки, т.е. из перестановки длины i-*1*. Таким образом, мы определили переходы –

из $D[i][j]$ переходим в $D[i-1][j-k]$ для *k=0. . . i-1* (т.е. динамика назад). Базой динамики является элемент $D[0][0] = 1$.

Задача D. Дириктивы include

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	4 с
Ограничение по памяти:	256 Мб

Многие языки содержат специальную директиву, которая вставляет содержимое одного файла внутрь другого. Часто такие директивы называются *include*. В этой задаче вам предстоит реализовать функциональность простейшего препроцессора, обрабатывающего эту инструкцию. Вам задан набор текстовых файлов. Строка файла вида “#include <имя-файла>” должна быть заменена на содержимое соответствующего файла (имя файла в директиве может быть как абсолютным так и относительным; относительный путь всегда начинается с имени файла или директории или спецпоследовательности “..”). В этой строке могут быть любые незначащие пробелы, не разрывающие элементы include и имя-файла. Если файла с именем имя-файла не существует, то эта строка должна быть просто вырезана из текста файла.

Формат входного файла

Входной файл состоит из последовательности описаний файлов. Каждое описание начинается со строки, содержащей путь до файла в формате файловых систем Windows. Путь задан абсолютно и не содержит пробелов и символов “\” идущих подряд. В качестве элемента пути может быть использована спецпоследовательность “..”, которая обозначает переход в родительскую директорию. Другие спецпоследовательности не используются. Имя любой директории и файла состоит из латинских букв, цифр и символа “.”. Сравнение имен файлов следует проводить без учета регистра. Далее идет содержимое файла. Последовательность вида “^Z” обозначает конец файла. Эта последовательность всегда записана на отдельной строке, то есть каждая строка файла, включая последнюю, завершается переводом строки. Входной файл содержит не менее одного описания. Размер файла не превосходит 40KB. Длина каждой строки файла не превосходит 1000 символов. Все имена файлов в тесте (включая те, что встречаются в директивах include) - это корректные абсолютные или относительные пути.

Формат выходного файла

Выведите содержимое первого файла после обработки препроцессором. Гарантируется, что размер вывода не превзойдет 400КБ. Если обработка директив приводит к циклическому процессу - выведите строку “Too long file”.

Пример

d.in	d.out
c:\ files\ first.txt # define MAX_N 1024 # include <second.txt> last line ^ Z c:\ FILES\ ..\ files\ second.txt included file # include <..\ files\ third.txt> ^ Z c:\ FILES\ THIRD.txt included file ^ Z	# define MAX_N 1024 included file included file last line
C:\ files\ first.txt # define MAX_N 1024 # include <second.txt> last line ^ Z c:\ FILES\ second.txt included file # include <..\ files\..\ files \ second.txt> ^ Z c:\ FILES\ THIRD.txt included file ^ Z	Too long file
c:\ 1\ a.txt # include "begin of a" # include <c:\ 2\ b.txt> # include end of a ^ Z c:\ 2\ ..\ 2\ b.txt begin of b #include <c.txt> end of b: # include <c.txt> ^ Z C:\ 2\ C.TXT oops ^ Z	# include "begin of a" begin of b oops end of b:# include <c.txt> # include end of a

Разбор задачи D. Директивы include

Представим описанную файловую систему в виде ориентированного графа, в котором вершинами являются файлы, а ребро проводится из одной вершины в другую, если имеется соответствующая директива include. Тогда, если в полученном графе есть циклы (возможно, петли), то следует вывести “Too long file”, в противном случае ответ всегда будет существовать.Остаётся только реализовать обработку директив include, а именно, научиться по текущему имени файла и относительному пути получать абсолютный путь. Это несложно, если представлять абсолютный путь как вектор строк, в котором первый элемент будет названием диска, последующие - названиями каталогов, и последний - именем файла.

Тогда каждый элемент относительного пути (имя каталога или “..”) можно воспринимать как добавление или удаление элемента из вектора соответственно. Вычислять сам ответ можно “лениво” — т.е. написать рекурсивную функцию, которой будет передаваться имя файла и которая будет возвращать обработанное содержимое файла, и запоми-нать результат в некоторой таблице. Проверку на ацикличность можно вставить в эту же функцию — если в процессе рекурсивных вызовов функция была вызвана от какого-либо файла, обработка которого ещё не завершена, то мы обнаружили цикл, и ответа не существует.

Задача E. Простецкие числа

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	4 с
Ограничение по памяти:	256 Мб

Число называется простецким, если его можно разбить на две части длиной не менее d цифр (каждая часть не может начинаться с 0) таких, что они обе являются простыми числами. Напомним, что простые числа – это такие натуральные числа, которые имеют ровно два различных делителя. Задана пара чисел d и n. Выведите наименьшее простецкое число не меньшее n.

Формат входного файла

Входной файл состоит из одного или более набора входных данных. Каждый набор записан в отдельной строке, содержащей пару натуральных чисел d и n, разделенных пробелом ($1 \le d \le 5$; $1 \le n \le 2*10^9$). Количество наборов входных данных в тесте не превосходит 5.

Формат выходного файла

Для каждого набора выведите искомое число на отдельной строке. Гарантируется, что ответ для любого набора не превзойдет $2 * 10^9$.

Пример

e.in	e.out
1 20 1 22 2 1 2 1234	22 22 1111 1311

Разбор задачи E. Простецкие числа

Переберём длину *len2* второй части: от *d* до *max(d, length(n))* — очевидно, большие значения рассматривать смысла нет. Дальше, во-первых, если *length(n) - len2 >= d*, то попробуем взять эту первую часть из числа *n*, и если она оказалась простой, то мы должны взять вторую часть числа *n* и найти следующее простое число после неё (при этом нужно учесть, что это число должно иметь длину, равную длине второй части, т.е. не начинаться с нуля). Во-вторых, мы можем взять первую часть больше, чем она в числе *n*, тогда вторую часть мы берём как можно меньшую (опять же, учитывая, что она должна иметь определённую длину). Из всех найденных ответов выбираем минимум и выводим его. Осталось только научиться достаточно быстро находить следующее простое число после заданного, но при данных ограничениях это можно было делать просто: увеличивать число на 1, пока не дойдём до простого числа (проверку на простоту также можно было делать самым наивным образом).

Задача F. Неточный поиск

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	4 с
Ограничение по памяти:	256 Мб

Задача поиска заданной подстроки в тексте имеет неоспоримое значение и ее решение реализовано в большинстве языков программирования в виде функциональности стандартной библиотеки. Однако, зачастую необходимо решать задачу неточного поиска. Назовем расстоянием между двумя строками *a* и *b* наименьшее количество операций, необходимое для того, чтобы

строку a перевести в строку b. Разрешенные операции: замена любого символа любым другим, вставка любого символа в произвольную позицию в строке и удаление произвольного символа. Например, расстояние между строками “кот” и “конь” равно 2. Ваша задача в заданном тексте найти подстроку, которая имеет расстояние до заданной не более d.

Формат входного файла

В первой строке входного файла записана строка в которой следует осуществлять поиск. Длина строки не менее 1 символа и не более $2 * 10^6$. Далее содержится строка, содержащая образец поиска. Его длина не менее 1 символа и не более 50 символов. Заданные строки содержат строчные и прописные буквы латинского алфавита и цифры. Последняя строка содержит целое число d ($0 \leq d \leq 50$) — наибольшее расстояние между образцом поиска и искомой подстрокой.

Формат выходного файла

Выведите два целых числа *start*, *length*, где *start* — позиция первого символа найденной подстроки, а *length* — ее длина. Нумерация символов в строке начинается с нуля. Если решений несколько, выведите любое.

Пример

f.in	f.out
thisisthetesttext tester 2	9 4
thesecondsampletestforproblemaboutthestrings pattern 4	12 5

Разбор задачи F. Неточный поиск

Решим эту задачу методом динамического программирования. Научимся вычислять динамику $D[i][j]$ — наименьшее расстояние между подстрокой, оканчивающейся в позиции i, и j-ым префиксом образца. Для строк будем пользоваться 1-индексацией, чтобы значения *i=0* или *j=0* соответствовали пустым подстрокам. Пусть мы уже вычислили элемент $D[i][j]$, тогда научимся выполнять переходы в другие элементы динамики. Первый переход — в элемент $D[i+1][j+1]$, здесь, если $S1[i+1] = S2[j+1]$, то мы выполняем этот переход бесплатно; если же $S1[i+1]! = S2[j+1]$, то за этот переход мы должны заплатить +1 (стоимость изменения одного

символа). Второй переход — в элемент $D[i+1][j]$, этот переход соответствует удалению одного символа (с номером *i+1*) из первой строки, поэтому за него мы должны заплатить +1. Последний переход — в элемент $D[i][j+1]$, этот переход соответствует удалению одного символа из второй строки, и за него мы также платим +1 к расстоянию редактирования.

Базой динамики является $D[i][0] = 0$, что означает, что мы может начать искомую подстроку в любой позиции, а начинаем мы с пустого совпадения с образцом, и стоимостью, равной нулю. Все остальные элементы динамики изначально равны бесконечности.

Если после вычисления динамики для какого-либо i выполняется $D[i][length(S2)] \leq d$, то мы нашли ответ – искомая подстрока оканчивается в позиции i. Чтобы найти позицию начала подстроки (и её длину), можно параллельно с построением динамики строить массив длин: $Len[i][j]$ – длина подстроки, соответствующей значению $D[i][j]$.

Стоить заметить, что при реализации создать массив D размером *length(S1)*length(S2)* не получится, поскольку он не поместится в память. Однако можно хранить на каждом шаге только массивы $D[i]$ и $D[i+1]$ (и, соответственно, $Len[i]$ и $Len[i+1]$), поскольку при фиксированном i их достаточно для выполнения переходов.

Задача G. Такси

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Управлять службой такси — сосвем не простое дело. Помимо естественной необходимости централизованного управления машинами для того, чтобы обслуживать заказы по мере их поступления и как можно быстрее, нужно также планировать поездки для обслуживания тех клиентов, которые сделали заказы заранее.

В вашем распоряжении находится список заказов такси на следующий день. Вам необходимо минимимизировать число машин такси, необходимых чтобы выполнить все заказы.

Для простоты будем считать, что план города представляет собой квадратную решетку. Адрес в городе будем обозначать парой целых чисел: x– координатой и y – координатой. Время, необходимое для того, чтобы добраться из точки с адресом *(a, b)* в точку *(c, d)*, равно |a-c|+|b-d| минут.

Машина такси может выполнить очередной заказ, либо если это первый ее заказ за день, либо она успевает приехать в назначенную точку из предыдущей конечной хотя бы за минуту до указанного срока. Обратите внимание, что выполнение некоторых заказов может окончиться после полуночи.

Формат входного файла

В первой строке входного файла записано число заказов M ($0 \leq M \leq 500$). Последующие M строк описывают сами заказы, по одному в строке. Про каждый заказ указано время отправления в формате hh:mm (в интервале с 00:00 по 23:59), координаты *(a, b)* точки отправления и координаты *(c, d)* точки назначения. Все координаты во входном файле неотрицательные и не превосходят 200. Заказы записаны упорядоченными по времени отправления.

Формат выходного файла

В выходной файл выведите единственное целое число — минимальное количество машин такси, необходимых для обслуживания всех заказов.

Пример

g.in	g.out
2 08:00 10 11 9 16 08:07 9 16 10 11	1
2 08:00 10 11 9 16 08:06 9 16 10 11	2

Разбор задачи G. Такси

Рассмотрим граф, в котором вершинами являются заказы. Если машина успевает отработать один заказ i, а потом успеть на заказ j, то проведем ребро из i в j. Тогда задача сводится к разбиению ациклического графа на множество непересекающихся путей. Это можно сделать, построив двудольный граф, в котором будет в два раза больше вершин с такой же матрицей смежностью. Найдем в этом графе максимальное паросочетание. Каждого ребро паросочетания будет некоторым ребром в одном из путей. Вершины первой доли, которые не входят в паросочетание, будут стартовыми вершинами путей. Аналогично, вершины первой доли — конечными вершинами путей. Таким образом, ответом будет N - M, где N — количество вершин, а M — размер максимального паросочетания.

Задача H. Великая стена

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

У короля Людовика двое сыновей. Они ненавидят друг друга, и король боится, что после его смерти страна будет уничтожена страшными войнами. Поэтому Людовик решил разделить свою страну на две части, в каждой из которых будет властвовать один из его сыновей. Он посадил их на трон в города A и B, и хочет построить минимально возможное количество фрагментов стены таким образом, чтобы не существовало пути из города A в город B.

Страну, в которой властвует Людовик, можно упрощенно представить в виде прямоугольника $m \times n$. В некоторых клетках этого прямоугольника расположены горы, по остальным же можно свободно перемещаться. Кроме этого, ландшафт в некоторых клетках удобен для строительства стены, в остальных же строительство невозможно.

При поездках по стране можно перемещаться из клетки в соседнюю по стороне, только если ни одна из этих клеток не содержит горы или построенного фрагмента стены.

Формат входного файла

В первой строке входного файла содержатся числа m и n ($1 \le m, n \le 50$). Во второй строке заданы числа k и l, где $0 \le k, l, k + l \le mn - 2$, k — количество клеток, на которых расположены горы, а l — количество клеток, на которых можно строить стену. Естественно, что на горах строить стену нельзя. Следующие k строк содержат координаты клеток с горами x_i и y_i, а за ними следуют l строк, содержащие координаты клеток, на которых можно построить стену — x_j и y_j. Последние две строки содержат координаты городов A (x_A и y_A) и B (x_B и y_B) соответственно. Среди клеток, описанных в этих $k + l + 2$ строках, нет двух совпадающих. Гарантируется, что $1 \le x_i, x_j, x_A, x_B \le m$ и $1 \le y_i, y_j, y_A, y_B \le n$.

Формат выходного файла

В первой строке выходного файла должно быть выведено минимальное количество фрагментов стены F, которые необходимо построить. В последующих F строках необходимо вывести один из возможных вариантов

застройки. Если невозможно произвести требуемую застройку, то необходимо вывести в выходной файл единственное число -1.

Пример

h.in	h.out
5 5 3 8 3 2 2 4 3 4 3 1 1 3 2 3 3 3 4 3 5 3 1 4 1 5 2 1 5 5	3 3 1 1 3 3 3

Разбор задачи H. Великая стена

Эта задача сводится к нахождению минимального разреза по вершинам. То есть необходимо удалить наименьшее количество вершин, чтобы перестал существовать путь между двумя заданными вершинами. Необходимо каждую вершину превратить две вершины и провести ребро пропускной способностью 1 между ними.

Таким образом задача сведется к нахождению классического наименьшего разреза.

Задача I. Разрез

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Найдите минимальный разрез между вершинами 1 и n в заданном неориентированном графе.

Формат входного файла

На первой строке входного файла содержится n ($1 \le n \le 100$) — число вершин в графе и m — количество ребер. На следующих m строках входного файла содержится описание ребер. Ребро описывается номерами вершин, которые оно соединяет, и его пропускной способностью (положительное целое число, не превосходящее десять миллионов), при этом никакие две вершины не соединяются более чем одним сегментом.

Формат выходного файла

На первой строке выходного файла должны содержаться количество ребер в минимальном разрезе и их суммарная пропускная способность. На следующей строке выведите возрастающую последовательность номеров ребер (ребра нумеруются в том порядке, в каком они были заданы во входном файле).

Пример

i.in	i.out
3 3 1 2 3 1 3 5 3 2 7	2 8 1 2

Разбор задачи I. Разрез

Задача на теорию, рассказанную на лекции. Необходимо реализовать алгоритм Эдмонса-Карпа.

Задача J. Испорченный паркет

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Пол в некоторой комнате размером $M \times N$ замощен паркетом. При этом некоторые плитки паркета оказались испорчены. Петя решил сделать ремонт в этой комнате, заменив только испорченные клетки. Придя в магазин, он обнаружил, что паркетные плитки бывают двух типов - размера 1×2, которые стоят A рублей (немного подумав, Петя понял, что плитки 1×2 можно поворачивать на 90 градусов, получая тем самым плитки 2×1) и размера 1×1, которые стоят B рублей. Разрезать плитку размера 1×2

на две, размера 1×1 Петя не может.

Определите, какая минимальная сумма денег нужна Пете, чтобы сделать ремонт.

Формат входного файла

Первая строка входного файла содержит 4 числа *N, M, A, B* ($1 \leq N, M \leq 300$, A, B - целые числа, по модулю не превосходящие 1000).

Каждая из последующих N строк содержит по M символов: символ "." (точка) обозначает неиспорченную плитку паркета, а символ "*" (звездочка) – испорченную.

В конце строк могут идти незначащие пробелы. В конце файла могут быть пустые строки.

Формат выходного файла

В выходной файл выведите одно число — минимальную сумму денег, имея которую можно заменить испорченные паркетины (и только их).

Пример

j.in	j.out
2 3 2 2 .** .*.	5

Разбор задачи J. Испорченный паркет

Если две плитки размером один на один дешевле, чем одна размером два на один, то можно все замостить единичными плитками. В противном случае необходимо положить как можно больше плиток размером два на один. Построим граф, вершинами которого будут испорченные клетки, а ребром будут соединены клетки, которые можно замостить одно плиткой один на два. Заметим, что это двудольный граф, в котором нужно найти максимальное паросочетание. Это можно сделать с помощью любого алгоритма нахождения потока или алгоритма.

День четвертый (21.02.2009 г.)
Контест Виталия Гольдштейна

Теоретический материал. Близость точек на плоскости

1. Нахождение максимального паросочетания в двудольном графе

Для нахождения максимального паросочетания необходимо построить сеть. Существующим в графе ребрам установим пропускную способность равную 1. Добавим новые вершины сток и исток s и t, соединим исток со всеми вершинами первой доли, а вершины второй доли соединим со стоком. Все добавленные ребра сделаем пропускной способностью 1. Найдем максимальный поток в полученной сети и получим максимальное паросочетание. Ребра, по которым течет поток будут входить в паросочетание, а другие нет.

2. Проверка единственности минимального разреза

Найдем максимальный поток. Найдем множество S вершин, достижимых из стока в остаточной сети, и множество T вершин, из которых достижим сток. Минимальный разрез единственен тогда и только тогда, когда $SUT = V$.

3. Пропускная способность вершины

В случае если требуется найти множество путей не пересекающихся по ребрам или проходящих по некоторым вершинам не более фиксированного количества раз, можно ввести понятие пропускной способности вершины.

В таком случае можно заменить одну вершину на две и провести ребро между ними необходимой пропускной способностью. Все входящие ребра будут входить в одну вершину, а выходящие ребра выходят из второй.

4. Блокирование наименьшего числа вершин, чтобы перестал существовать путь из вершины s в вершину t

Эта задача похожа на поиск наименьшего разреза. Однако здесь можно блокировать вершины, поэтому введем пропускную способность вершин равную 1. Размер наименьшего разреза и будет наименьшим количеством вершин.

Можно так же ввести стоимость блокирования вершин и ребер. Решение не изменится, просто пропускная способность вершин и ребер будет равна стоимости блокирования вершин и ребер соответственно.

5. Ориентация ребер, чтобы граф стал Эйлеровым

Пусть задан граф, некоторые ребра, в котором ориентированные, а некоторые нет. Необходимо ориентировать все ребра так, чтобы граф был Эйлеровым. Напомним, что ориентированный граф Эйлеров тогда и только тогда, когда количество входящих ребер равно количеству исходящих для каждой вершины.

Посчитаем для каждой вершины баланс входящих и исходящих ребер для каждой вершины. Если баланс отрицательный $d < 0$ (то есть количество входящих ребер больше, чем исходящих) необходимо сделать - d ребер исходящими, в противном случае нужно сделать d ребер входящими.

Добавим в граф две вершины s и t. Во все вершины с отрицательным балансом проведем ребро - d из стока, а из вершин с положительным балансом проведем ребра d в сток. Найдя в полученной сети поток, ориентируем ребра вдоль потока, идущего по ним. В таком случае баланс каждой вершины станет равен 0. Но поток может пройти не по всем ребрам, поэтому некоторые ребра останутся неориентированными.

Оставшиеся ребра можно ориентировать вдоль Эйлерового пути в каждой компоненте связности. Если количество ребер инцидентных с каждой вершиной в первоначальном графе четно, то Эйлеров цикл найдется в каждой компоненте. Если это не так, то с самого начала можно сказать, что ориентация невозможна.

Задачи и разборы

Задача A. Мороженое

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

По дороге в школу Петя любит забегать в киоск и покупать себе мороженое. Однако при этом он часто опаздывает в школу. Неожиданно Петя понял - он просто ходит не по кратчайшему пути!

Помогите Пете победить опоздания. Город можно представить как n перекрестков, соединенных m улицами, про каждую улицу известна ее длина. Дом Пети находится на перекрестке a, школа - на перекрестке b, а киоск с мороженым - на перекрестке c. По пути в школу Петя никогда не проходит через один перекресток дважды.

Формат входного файла

Первая строка входного файла содержит числа n и m ($3 \leq n \leq 30000, 0 \leq m \leq 50000$). Вторая строка содержит три различных числа - a, b и c. Следующие m строк содержат по три целых числа x_i, y_i и l_i - номера перекрестков, соединенных улицей и ее длину (длина - целое положительное число, которое не превышает 10^4).

Формат выходного файла

Если путь из дома Пети до школы, проходящий через перекресток с мороженым и не проходящий по одному перекрестку два раза, существует, выведите на первой строке выходного файла два целых числа k и l - количество улиц, которые проходит Петя в оптимальном пути и длину пути. На второй строке выведите номера перекрестков в том порядке, в котором их посещает Петя. В противном случае выведите -1 на первой строке выходного файла.

Пример

a.in	a.out
5 6 1 5 3 1 2 1 2 3 4 2 4 1 3 4 2 5 1 1 4 5 2	4 9 1 2 3 4 5

Разбор задачи A. Мороженое

Задачу можно переформулировать следующим образом. Найдите два не пересекающихся пути от киоска с мороженым до школы и дома Пети соответственно. Суммарная длина пути должна быть минимальна. Это можно сделать с помощью поиска потока минимальной стоимости. Необходимо запустить две итерации поиска кратчайшего пути.

Задача B. Коллекционирование монет

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

В клубе коллекционеров монет состоят n человек. В настоящее время клубом проводится конкурс - кто первым соберет полную коллекцию монет XIX века. По требованиям клуба, эта коллекция должна состоять из m различных типов монет, причем, можно иметь больше одной монеты каждого типа, но обязательно хотя бы одну.

Вася решил подойти к этому вопросу основательно и хочет рассчитать оптимальную стратегию обмена монет. Он знает, что любой коллекционер с радостью обменяет любую свою монету a на любую Васину монету b, но только в одном случае: если у этого коллекционера больше одной монеты вида a, но еще нет ни одной монеты серии b. Ни на какие другие обмены коллекционеры не согласны. Сам Вася, в отличие от других, ставит глобальную оптимизацию выше локальной, и при желании может нарушать это правило.

Вася хочет с помощью таких обменов набрать как можно больше различных типов монет, чтобы облегчить себе в будущем задачу победы в конкурсе. Помогите ему рассчитать оптимальную стратегию обмена монетами!

Формат входного файла

В первой строке входного файла заданы два числа n и m ($1 \leq m$, $n \leq 100$). Далее следуют n строк по m чисел в каждой: j-е число i-й строки - это количество монет j-го типа у i-го коллекционера. Все эти числа неотрицательны и не больше 1000. Вася имеет номер 1.

Формат выходного файла

В первой строке выходного файла выведите максимальное количество различных типов монет, которые можно собрать. Далее выведите любую из последовательностей обменов, приводящую к такому количеству различных типов. Каждый из обменов выводится в формате x_i, u_i, v_i, где x_i - номер коллекционера, с которым следует обменяться, u_i - номер монеты, которую Вася от него получает, а v_i - номер монеты, которую Вася ему отдает.

Пример

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

Разбор задачи B. Парадокс дней рождения

Рассчитаем сначала, какова вероятность $p(n)$ того, что в группе из n человек дни рождения всех людей будут различными. Если $n > 365$, то в силу принципа Дирихле вероятность равна нулю. Если же $n \leq 365$, то будем рассуждать следующим образом. Возьмём наугад одного человека из группы и запомним его день рождения. Затем возьмём наугад второго человека, при этом вероятность того, что у него день рождения не совпадёт с днем рождения первого человека, равна $1-1/365$. Затем возьмём третьего человека, при этом вероятность того, что его день рождения не совпадёт с днями рождения первых двух, равна $1 - 2/365$.

Рассуждая по аналогии, мы дойдём до последнего человека, для которого вероятность несовпадения его дня рождения со всеми предыдущими будет равна $1 - (n - 1)/365$. Перемножая все эти вероятности, получаем вероятность того, что все дни рождения в группе будут различными:

$$\overline{p(n)} = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{365}\right)$$

Тогда вероятность того, что хотя бы у двух человек из n дни рождения совпадут, равна

$$p(n) = 1 - \overline{p(n)}$$

Задача C. Уголки

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	4 с
Ограничение по памяти:	64 Мб

Монохромный графический файл имеет разрешение $n \times m$ пикселей. Каждый пиксель имеет либо белый, либо черный цвет. Компонента связности черных пикселей - это каждое наибольшее по включению множество черных пикселей, такое, что каждый пиксель множества достижим из любого другого по черным пикселям при перемещениях вправо, влево, вверх или вниз. Например, в первом тесте изображено четыре компоненты связности черных пикселей. Компонента называется уголком, если она состоит из двух перпендикулярных отрезков, пересекающихся в концах. Например, в первом тесте изображено два уголка. Найдите количество уголков на заданном рисунке.

Формат входного файла

Входной файл содержит одно или более изображение. В первой строке описания изображения записаны два натуральных числа n, m $(1 \leq n \leq 50; 1 \leq m \leq 50)$. Далее содержится n, строк, каждая по m символов. Символ “ . ” соответствует пикселю белого цвета, а символ “ * ” - черного. Количество изображений в файле не превосходит 50.

Формат выходного файла

Для каждого изображения выведите количество уголков на картинке в отдельной строке.

Пример

c.in	c.out
9 14 ....****.. .*.........*.. .*.........*.. .*.........*.. .****....***.. .....***....*. .....*........ .....*........ 8 8 *******. *......* *.****.* *.*....* *...*..* *...*..* *......* .*******	2 3

Разбор задачи C. Уголки

Переберем клетку (x, y) - вершину уголка. Проверим, что эта клетка - черная, что у нее ровно две соседних черных клетки, и что эти соседние клетки образуют прямой угол. Теперь осталось проверить, что продолжения соседних клеток являются отрезками без ответвлений, и, если это так, прибавить 1 к ответу.

Задача D. Встроенная очередь команд

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	4 с
Ограничение по памяти:	64 Мб

Встроенная очередь команд (Native Command Queuing, NCQ) - это технология, использующаяся в SATA-устройствах для повышения быстродействия. NCQ позволяет устройству накапливать запросы, оптимизировать

порядок их выполнения с учётом внутренней архитектуры устройства (минимизация количества перемещений головок, простоя в ожидании нужного сектора на треке). В этой задаче мы пренебрежем значением ряда факторов и будем учитывать только лишь время перемещения считывающей головки SATA-устройства. Изначально будем считать, что головка расположена в позиции 0. Скорость ее перемещения равна 1 (одна позиция в миллисекунду). Вам заданы команды в виде (x_i, t_i), где x_i - позиция из которой надо прочесть (или записать) информацию, а t_i - время поступления запроса. Запросы не обязательно должны быть обработаны в порядке их поступления. Главное, чтобы головка устройства заняла позицию x_i в момент времени t_i или позже. Время чтения/записи будем считать пренебрежимо малым. Найдите наименьшее время, за которое устройство обработает все поступившие команды.

Формат входного файла

В первой строке входного файла записано целое число n $(1 \leq n \leq 2000)$ - количество поступивших команд. Далее перечислены сами команды - по одной в строке. Команды задаются парами целых чисел $x_i, t_i (-10^6 \leq x_i \leq 10^6; 0 \leq t_i \leq 10^6)$.

Формат выходного файла

Выведите единственное число – наименьшее время обработки всех команд.

Примеры

d.in	d.out
2 4 5 1 7	8
1 1 1000	1000

Разбор задачи D. Распределение

Отсортируем все запросы в порядке возрастания x координаты. Все дальнейшие рассуждения будем вести относительно вновь полученного порядка. Для начала заметим одно важное свойство оптимального решения. Рассмотрим простой пример. Пусть у нас есть 3 запроса: $q_1 = (2, 10), q_2 = (3, 20), q_3 = (1, 30)$.

Очевидно, что одним из оптимальных решений является выполнение этих запросов в порядке (q_1, q_2, q_3). Однако запрос q_1 мы можем выполнить не первым, а в тот момент, когда головка перемещается от x_2 к x_3, и при этом ответ не ухудшится. Можно обобщить данное наблюдение на N запросов. А именно, пусть есть оптимальное решение $(q_{i1}, q_{i2}, ..., q_{in})$ и в нем существует подпоследовательность из трех запросов (обозначим их через q_{j1}, q_{j2} и q_{j3}), таких, что верны неравенства $t_{j1} \leq t_{j2} \leq t_{j3}$ и ($x_{j3} \leq x_{j1} \leq x_{j2}$ или $x_{j3} \leq x_{j1} \leq x_{j2}$). В этом случае, по аналогии с примером, q_{j1} можно будет выполнить между выполнением q_{j2} и q_{j3}, не ухудшив ответ. Если применять данное преобразование ответа до тех пор, пока в нем существует подпоследовательность с указанными свойствами, то получим решение с таким же временем выполнения всех запросов, но в новом решении номера еще не выполненных запросов всегда будут целиком покрывать некоторый отрезок $[L, R]$. А, значит, теперь можем применить для решения задачи динамическое программирование. Пусть тройка чисел (L, R, pos) описывает следующее состояние - имеем отрезок $[L, R]$, головка находится в точке xR, если $pos = 1$ и в точке xL в противном случае. Причем, все запросы с номерами от L до R еще не обработаны, кроме запроса, в позиции которого находится головка (считаем, что этот запрос обработался, только что). Значением $d[L][R][pos]$ для состояния является минимальное время, за которое данное состояние достигается. Рассмотрим переходы для состояния $(L, R, 0)$. Так как запрос с номером L обработан только что, то переход будет в отрезок $[L + 1, R]$. Далее, перебираем, в какой конец вновь полученного отрезка перемещаем головку. Рассмотрим случай для перемещения в правый. Обозначим время, за которое головка совершит выбранное перемещение через $moveTime$. Тогда момент времени, к которому головка закончит передвижение, будет равен $d[L][R][0] + moveTime$. Может случиться так, что к этому моменту запрос еще не поступил, т.е. $d[L][R][0] + moveTime < tR$. В этом случае нужно дождаться поступления запроса, не перемещая головку. Таким образом, получим следующий переход: $d[L + 1][R][1] = min(d[L + 1][R][1], max(d[L][R][0] + moveTime, tR))$. Остальные переходы выполняются аналогично.

Ответом на задачу является минимум по всем i величин $d[i][i][0]$.

Задача E. Клуб “Двоичный кот”

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	4 с
Ограничение по памяти:	64 Мб

Охрана клуба “Двоичный кот” никогда не дремлет. Охранник записывает имя каждого посетителя и время, когда он вошел в клуб или вышел из него. Пометок о том, какому событию соответствует эта запись он не ставит, так как понимает, что нечетная запись для данного человека обозначает, что он вошел в клуб, а четная, что вышел из клуба. Сотрудники специального отдела государственной службы безопасности хотят получить сведения о посетителях, которые находились в клубе в заданные моменты времени. Помогите охране клуба предоставить эту информацию.

Формат входного файла

В первой строке входного файла записана пара целых чисел n и m ($1 \le n \le 1000; 1 \le m \le 1000$), где n - количество записей в журнале охраны, а m - количество запросов спецслужбы. Далее в n строках содержатся описания записей в формате “hh:mm:ss имя”, где hh:mm:ss - время в стандартном формате (ровно 8 символов), а “имя” – это имя человека зашедшего или вышедшего из клуба. Имя состоит из строчных или прописных букв латинского алфавита и имеет длину от 1 до 16 символов включительно. Записи расположены в порядке неубывания времен событий. Далее содержится m строк. Каждая строка содержит запись вида “hh:mm:ss” задающую момент времени, которым интересуются сотрудники госбезопасности. Все времена во входном файле ограничены одними сутками.

Формат выходного файла

Выведите m строк в выходной файл. Каждая строка должна содержать список людей, которые находятся в клубе в соответствующий момент времени. Список должен начинаться с количества людей, а затем должна следовать последовательность имен. Элементы списка следует разделять пробелами. Имена в списках можно выводить в любом порядке. Так как сотрудники госбезопасности не хотят упустить кого-либо из списка, то если именно в эту секунду посетитель зашел в клуб, считайте, что он находится в клубе в эту секунду. Если посетитель именно в эту секунду вышел из клуба, то следует считать, что он его еще покинуть не успел, то есть посетитель еще находится в клубе.

Примеры

e.in	e.out
4 2 00:00:19 Mike 00:00:26 Kate 02:42:11 Mike 02:42:11 Kate 00:00:20 04:00:00	1 Mike 0
7 3 06:00:00 Athos 06:00:00 Porthos 06:00:00 Aramis 07:00:00 Athos 07:00:00 Porthos 07:00:00 Aramis 07:10:00 dArtagnan 06:00:00 06:30:00 07:00:00	3 Aramis Athos Porthos 3 Aramis Athos Porthos 3 Aramis Athos Porthos

Разбор задачи E. Клуб “Двоичный кот”

Для начала для каждого человека найдем набор временных промежутков, когда он находился в клубе. Это удобно делать, если завести массив times, в котором будет храниться время последнего входа человека в клуб, или -1, если данный человек сейчас не в клубе. Тогда при обработке очередной записи охраны делаем следующие действия: если в массиве times для данного человека стоит -1, то фиксируем новое время входа, иначе в набор временных промежутков добавляем отрезок [times[<номер человека>], <время записи>], после чего в times[<номер человека>] ставим -1. После обработки записей охраны, для всех людей, которые еще находятся в клубе (т.е. для которых times[<номер человека>] $\neq -1$) необходимо добавить в набор временных промежут-ков отрезок ([times[<номер человека>], ∞]). Теперь приступим к обработке запросов. Для каждого запроса в ответ попадут те люди, для ко-торых в наборе временных промежутков существует отрезок [l, r], такой, что $l \leq$ <время запроса> и <время запроса> $\leq r$.

Задача F. Балансировка нагрузки

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	4 с
Ограничение по памяти:	64 Мб

Процессоры многопроцессорной системы соединены в цепь последовательно, один за одним. Они пронумерованы целыми числами от 1 до n. Каждый из них содержит некоторый набор заданий на исполнение, i- ый процессор содержит t_i заданий. Система будет сбалансированной, если каждый процессор будет содержать одинаковое количество заданий. Балансировка нагрузки происходит раундами. Каждый раунд каждый процессор может "передать" не более одного задания каждому своему соседу. Соседями процессора i являются процессоры с номерами $i-1$ и $i+1$, крайние процессоры имеют лишь по одному соседу. Найдите наименьшее количество раундов, необходимое для балансировки системы.

Формат входного файла

В первой строке входного файла записано целое число n - количество процессоров ($1 \leq n \leq 5000$). Вторая строка содержит последовательность t. Последовательность имеет длину n и содержит целые числа, каждое из которых от 0 до 50000 включительно.

Формат выходного файла

Выведите наименьшее количество раундов, необходимое для балансировки системы. Если сумма значений последовательности t не кратна n, то решения не существует. В этом случае выведите -1.

Примеры

f.in	f.out
4 0 3 0 1	1
4 2 2 4 4	2
5 3 1 4 1 6	3

Разбор задачи F. Балансировка нагрузки

Очевидно, что в оптимальном решении между двумя соседними про-

цессорами все передачи заданий идут в одну и ту же сторону. Тогда для каждой пары соседних процессоров можно посчитать число передач между ними (оно равно количеству лишних заданий с одной стороны от этой пары).

Ответом будет максимум этой величины по всем таким парам. Может показаться, что если есть процессоры, у которых изначально нет заданий, то ответ будет не таким, так как для этих процессоров передачи начнутся с задержкой. Но нетрудно показать, что это не повлияет на правильный ответ.

Задача G. Охлаждение реактора

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Известная террористическая группа под руководством знаменитого террориста Бен Гадена решила построить атомный реактор для получения оружейного плутония. Вам, как компьютерному гению этой группы, поручили разработать систему охлаждения реактора.

Система охлаждения реактора представляет собой набор труб, соединяющих узлы. По трубам течет жидкость, причем для каждой трубы строго определено направление, в котором она должна по ней течь. Узлы системы охлаждения занумерованы от 1 до N. Система охлаждения должна быть спроектирована таким образом, чтобы для каждого узла за единицу времени количество жидкости, втекающей в узел, было равно количеству жидкости, вытекающей из узла. То есть если из i-го узла в j-ый течет f_{ij} единиц жидкости за единицу времени (если из i в j нет трубы, то положим $f_{ij} = 0$), то для каждого узла i должно выполняться:

$$\sum_{j=1}^{N} f_{ij} = \sum_{j=1}^{N} f_{ji}$$

У каждой трубы имеется пропускная способность c_{ij}. Кроме того, для обеспечения достаточного охлаждения требуется, чтобы по трубе протекало не менее l_{ij} единиц жидкости за единицу времени. То есть для трубы, ведущей из i-го узла в j-ый должно выполняться $l_{ij} \le f_{ij} \le c_{ij}$.

Вам дано описание системы охлаждения, выясните, каким образом можно пустить жидкость по трубам, чтобы выполнялись все указанные условия.

Формат входного файла

Первая строка входного файла содержит числа N и M - количество узлов и труб ($1 \leq N \leq 200$). Следующие M строк содержат описание труб. Каждая строка содержит четыре целых числа i, j, l_{ij} и c_{ij}. Любые два узла соединены не более чем одной трубой, если есть труба из i в j, то нет трубы из j в i, никакой узел не соединен трубой сам с собой, $0 \leq l_{ij} \leq c_{ij} \leq 10^5$.

Формат выходного файла

Если решение существует, выведите на первой строке выходного файла слово YES. Затем выведите M чисел – количество жидкости, которое должно течь по трубам, числа должны быть выведены в том порядке, в котором трубы заданы во входном файле. Если решения не существует, выведите NO.

Примеры

g.in	g.out
4 6 1 2 1 2 2 3 1 2 3 4 1 2 4 1 1 2 1 3 1 2 4 2 1 2	NO
4 6 1 2 1 3 2 3 1 3 3 4 1 3 4 1 1 3 1 3 1 3 4 2 1 3	YES 1 2 3 2 1 1

Разбор задачи G. Охлаждение реактора

Это задача на нахождение циркуляции в сети с заданными нижними ограничениями на ребра. Если по ребру (u, v) должен проходить поток в отрезке [l, r], то в перестроенной сети будет три ребра: (u, v, r - l), (s, v,

l), (u, t, l). s, t - дополнительно введенные сток и исток соответственно. Фактически мы пропускаем по ребру необходимый минимальный поток, после чего балансируем его так, чтобы получить циркуляцию.

Задача H. Пешеходные зоны против кольцевых

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	4 с
Ограничение по памяти:	64 Мб

В городе C. очень давно не проводились дорожные реформы. Наконец, мэр города принял решение положить конец пробкам в городе. Для этого он решил все дороги в городе сделать односторонними. Более того, весь город будет покрыт кольцевыми автомобильными дорогами. Односторонняя дорога является частью кольцевой, если из ее конца можно проехать в ее начало, двигаясь по дорогам города (в разрешенных направлениях). Некоторые дороги невозможно сделать частью кольцевой, такие дороги решено превратить в пешеходные зоны. Выведите суммарную длину пешеходных зон в городе после реформы. Дороги заданы отрезками на плоскости, которые, возможно имеют общие части. Наличие общей части у двух или более дорог не обозначает существование параллельных дорог “рядом”, а лишь то, что одна часть представлена в разных отрезках на плоскости.

Формат входного файла

В первой строке входного файла содержится целое число n $(1 \leq n \leq 40)$, где n – количество дорог в городе. Далее в n строках заданы отрезки парами координат своих концов. Все отрезки имеют положительную длину. Координаты не превосходят 100 по абсолютной величине.

Формат выходного файла

Выведите суммарную длину всех пешеходных зон, полученных в результате реформы. Ответ выводите с точностью 5 знаков после десятичной точки.

Примеры

h.in	h.out
4 0 0 10 0 10 5 5 5 5 5 5 0 10 0 10 5	5
4 0 3 2 3 1 3 3 3 3 3 3 0 0 3 4 -1	1.41421
2 0 0 6 8 0 0 6 8	10

Разбор задачи H. Пешеходные зоны против кольцевых

Построим неориентированный граф, в котором вершинами будут точки пересечения отрезков и концы всех отрезков. Между двумя вершинами будет ребро, только если есть содержащий их отрезок, и на этом отрезке между ними нет других вершин. Очевидно, что мы не сможем включить ни в какой цикл только мосты получившегося графа. То есть нам остается просто найти все мосты и прибавить к ответу их длины. Всего вершин в графе - O(n2) (на самом деле, с константой около 2), ребер - O(n3) (из каждой вершины O(n) ребер). Строим мы граф за O(n3log(n)), находим мосты за O(n3) (линейно количеству ребер). Так, асимптотика решения - O(n3log(n)).

Задача I. Кони

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Шахматным конем называется фигура, которая за один ход на клетчатом поле перемещается на 2 клетки в одном из 4 направлений (влево, вправо, вверх или вниз) и на 1 клетку в перпендикулярном. Будем говорить, что фигура контролирует клетку, если она может достичь ее за любое число

ходов. Поле также может иметь бесконечную длину и/или ширину. Это означает, что поле имеет бесконечное число клеток, например поле $INF \times M$ будет представлять собой полосу бесконечной длины шириной M, а поле $INF \times INF$ - абсолютно бесконечное поле, не имеющее границ. Дано поле $N \times M$. ($1 \leq N, M \leq 1000$) или бесконечность(INF)), найдите минимальное число коней, для того, чтобы контролировать все поле.

Формат входного файла

В первой строке входного файла содержится число N – длина шахматного поля или слово *INF* (заглавными латинскими буквами), если поле имеет бесконечную длину. Во второй строке содержится число M – ширина шахматного поля или слово *INF* (заглавными латинскими буквами), если поле имеет бесконечную ширину.

Формат выходного файла

Выведите единственное число – ответ на поставленную задачу. Если для достижения результата требуется бесконечное количество коней, то выведите INF.

Примеры

i.in	i.out
8 8	1

Разбор задачи I. Кони

Пусть у нас ширина не больше длины (в противном случае, поменяем длину и ширину местами). Далее рассмотрим все случаи, которые только могут быть.

Полоса шириной 1, и любой длины. Поскольку конь не может перемещаться по такому полю, то необходимо поставить коня в каждую клетку - то есть ответ есть длина нашего поля.

Полоса шириной 2. Необходимое число коней - 4. На такой полосе, конь может перемещаться в клетку с той же четностью по длине (но отличающейся на 2), и различающейся точности по ширине. Таким образом, поставив 4х коней в клетки с координатами (1, 1) (1, 2) (2, 1) (2, 2) мы сможем оккупировать все клетки.

Поле 3 на 3. Необходимо 2 коня - одного ставим в центр, другого куда-нибудь еще.

Остальные случаи. Во всех таких случаях поле содержит в себе прямоугольник 3x4, который можно обойти одним конем. Расширив данное поле, мы сможем обойти и его. Ответ - 1 конь.

Задача J. Сложите многоугольник

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	4 с
Ограничение по памяти:	64 Мб

Задан набор отрезков. Какое наименьшее число отрезков надо удалить из набора, чтобы из оставшегося числа отрезков можно было сложить многоугольник, используя отрезки в качестве его сторон. Многоугольник должен иметь ненулевую площадь.

Формат входного файла

В первой строке входного файла записано целое число $n(3 \leq n \leq 50)$, где n – количество отрезков в наборе. Вторая строка содержит длины отрезков – последовательность длины n целых чисел, каждое из которых от 1 до 10^6 включительно.

Формат выходного файла

Выведите наименьшее количество отрезков, которых надо удалить из заданного набора, чтобы из оставшегося числа отрезков было возможно сложить многоугольник. Если из любого поднабора отрезков сложить многоугольник невозможно, выведите -1.

Примеры

j.in	j.out
4 1 1 1 100	1
3 1 2 3	-1

Разбор задачи J. Сложите многоугольник

Из произвольного набора отрезков можно сложить невырожденный многоугольник тогда и только тогда, когда длина наибольшего из отрезков строго меньше суммы длин остальных. Из этого следует, что если мы

взяли отрезок длины L в качестве одной из сторон, то если возьмем еще и все отрезки длины $\leq L$, то многоугольник не перестанет существовать, а ответ только улучшится. Значит, будем искать оптимальное множество следующим образом. Переберем отрезок, который будет максимальной по длине стороной в многоугольнике. Добавим к нему все отрезки, у которых длина не превосходит длины выбранного отрезка. После этого проверим, можно ли из полученного множества составить многоугольник, и если это возможно, то проверим построенный ответ на оптимальность.

День пятый (22.02.2009 г.) Контест Михаила Медведева

Об авторе...

Медведєв Михайло Геннадійович, доцент факультету кібернетики Київського національного університету імені Тараса Шевченка. Член журі Всеукраїнських та міжнародних студентських олімпіад з програмування. Тренер команд КНУ на чемпіонаті світу з програмування (1997 – 2003 рр.), тренер команд Києво - Могилянської академії (з 2003 р.). З 2003 року керівник студентських команд України з програмування, що приймають участь у півфінальному, південно-східному європейському регіоні. За роботу з обдарованою молоддю має почесні грамоти міністерства освіти і науки України (2003, 2008 рр.) та подяки (2005, 2006 рр.). Відмінник освіти України (2005 р.). Володар гранту Президента України (2007 р.). Автор багатьох статей з алгоритмів. Автор збірника задач з програмування (2004 р.).

Основні досягнення команд:

- 9 місце (бронзова медаль КНУ ім. Т.Шевченка) на фіналі 2003;
- 4 місце, Topcoder Component Design Competition 2006 (Сліпецький Ростислав, НаУКМА);
- 3 місце, Topcoder Component Design Competition 2007 (Сліпецький Ростислав, НаУКМА).

Вероятность.

Вероятностью будем называть меру достоверности случайного события.

Вероятностным пространством называется тройка (Ω, F, P), где

Ω - множество элементарных событий (исходов);

F - сигма-алгебра подмножеств, называемых случайными событиями;

P - вероятностная мера (или вероятность), такая что $P(\Omega) = 1$.

Пример. Рассмотрим эксперимент с бросанием монеты. Вероятностное пространство определим так:

$\Omega = \{0, 1\}$;

$F = \{\{0\}, \{1\}, \{0, 1\}, \varnothing\}$;

$P(\{0\}) = \frac{1}{2}, P(\{1\}) = \frac{1}{2}, P(\{0, 1\}) = 1, P(\{\varnothing\}) = 0.$

Условная вероятность. Для каждого события A и возможного события B, для которого $P(B) > 0$, условную вероятность события A при условии B определим как $P_B(A) = P(AB)/P(B)$

Формула полной вероятности. Пусть известны:

а) вероятности $P(B_i) = \beta_i$ нескольких исключающих друг друга условий B_i, одно из которых с достоверностью выполняется;

б) условные вероятности $P_{B_i}(A) = a_i$ события A при условии, что выполняется B_i;

Тогда вероятность наступления события A равна

$$P(A) = \sum_{i=1}^{n} P(B_i) \cdot P_{B_i}(A) = \sum_{i=1}^{n} \beta_i a_i$$

Формула Байеса. Пусть известны:

а) вероятности $P(B_i) = \beta_i$ возможных исключающих друг друга предположений B_i;

б) условные вероятности $P_{B_i}(A) = a_i$ события A при условии, что верно предположение B_i;

Тогда условная вероятность того, что верно предположение B_i при условии реализуемости события A равна

$$P_A(B_i) = \frac{P(B_i) \cdot P_{B_i}(A)}{\sum\limits_{j=1}^{n} P(B_j) \cdot P_{B_j}(A)} = \frac{\beta_i a_i}{\sum\limits_{j=1}^{n} \beta_j a_j}$$

Пример. На склад поступило 1000 подшипников. Из них 200 изготовлены на первом заводе, 460 на втором и 340 на третьем. Вероятность того, что подшипник окажется нестандартным, для первого завода равна 0,03, для второго 0,02, для третьего 0,01.

а) Какова вероятность того, что случайно выбранный подшипник является нестандартным?

б) Взятый наудачу подшипник оказался нестандартным. Какова вероятность того, что он изготовлен первым заводом?

Решение. Обозначим через H_1, H_2, H_3 вероятности того, что случайно выбранный подшипник изготовлен соответственно первым, вторым или третьим заводом. Они равны $P(H_1) = \frac{200}{1000} = 0{,}2$, $P(H_2) = \frac{460}{1000} = 0{,}46$, $P(H_3) = \frac{340}{1000} = 0{,}34$

Пусть A – событие, состоящее в том, что взятый подшипник нестандартный. Из условия задачи следует, что

$p_1 = P_{H_1}(\text{A}) = 0{,}03$, $p_2 = P_{H_2}(\text{A}) = 0{,}02$, $p_3 = P_{H_3}(A) = 0,01$

По формуле полной вероятности

$P(A) = P(H_1) * P_{H_1}(A) + P(H_2) * P_{H_2}(A) + P(H_3) * P_{H_3}0 =$
$= 0{,}2 * 0{,}03 + 0{,}46 * 0{,}2 + 0{,}34 * 0{,}01 = 0{,}006 + 0{,}0092 + 0{,}0034 =$
$= 0{,}0186$

Найдем $P_A(H_1)$ – вероятность того, что подшипник, оказавшийся нестандартным, изготовлен первым заводом. По формуле Бейеса имеем

$$P_A(H_1) = \frac{P(H_1)p_1}{p(H_1)p_1 + P(H_2)p_2 + P(H_3)p_3} =$$

$$= \frac{0,2 \cdot 0,03}{0,2 \cdot 0,03 + 0,46 \cdot 0,02 + 0,34 \cdot 0,01} \approx 0,32$$

Таким образом, вероятность гипотезы, что подшипник изготовлен первым заводом, изменилась после того, как стало известно, что он нестандартен.

Пример. В игорном клубе половина игроков честные, половина - шулеры. Вероятность вытащить из колоды короля равна 1/8. Для шулера эта вероятность равна 1. Сидящий перед вами игрок вытаскивает из колоды короля с первого раза. С какой вероятностью перед вами шулер?

Решение. Пусть событие A заключается в том, что из колоды вытянут король, B – в том, что игрок шулер. Тогда событие $\overline{B}$ заключается в том, что игрок честный, и $P(A\,|B) = 1$, $P(A\,|\overline{B}) = 1/8$. Если взять первого попавшегося игрока, то он вытянет короля с вероятностью

$$p(A) = p(B) \cdot p(A\,|B) + p(\overline{B}) = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{8} = \frac{9}{16}$$

Обозначим через X событие, заключающееся в том, что игрок, вытянувший короля, – шулер. Тогда

$$p(X) = \frac{p(B) \cdot p(A\,|B)}{p(A)} = \frac{p(B) \cdot p(A\,|B)}{p(B) \cdot p(A\,|B) + p(\overline{B}) \cdot p(A\,|\overline{B})} = \\ = \frac{\frac{1}{2} \cdot 1}{\frac{9}{16}} = \frac{8}{9}$$

Случайной величиной называется измеримая функция, заданная на некотором вероятностном пространстве. Вероятностное поведение случайной величины полностью описывается ее распределением.

Случайная величина ζ имеет **распределение Бернулли**, если она принимает всего два значения: 1 и 0 с вероятностями p и $q = 1 - p$ соответственно. Событие $\zeta = 1$ соответствует успеху, а $\zeta = 0$ – неудаче. Таким образом, $P(\zeta = 1) = p, P(\zeta = 0) = q$.

Пусть $X_1, X_2, ..., X_n$ – конечная последовательность независимых случайных величин с распределением Бернулли, то есть $P(X_i = 1) = p, P(X_i = 0) = q, i = 1, 2, ..., n$.

Построим случайную величину $Y = \sum\limits_{i=1}^{n} X_i$. Тогда число единиц (успехов) в последовательности имеет **биномиальное распределение**, причем $P(Y = k) = C_n^k p^k q^{n-k}$

Если одновременно устремить число опытов n к бесконечности, а вероятность p к нулю, причем их произведение сохраняет постоянное значение $np = \zeta$, то предельное свойство биномиального распределения можно записать в виде

$$\lim_{n\to\infty} C_n^k p^k q^{n-k} = \frac{\lambda^k}{k!} e^{-\lambda}$$

Распределение Пуассона моделирует случайную величину, представляющую собой число событий, произошедших за фиксированное время, при условии, что данные события происходят с некоторой фиксированной средней интенсивностью и независимо друг от друга. Если через λ обозначить интенсивность событий случайной величины Y, то

$$P(Y = k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

Пример. Устройство состоит из 1000 элементов, работающих независимо друг от друга. Вероятность отказа любого элемента в течение времени T равна 0,002. Найти вероятность того, что за время T откажут ровно три элемента.

Решение. Так как по условию n = 1000 достаточно велико, а p = 0,002 мало, то можно воспользоваться распределением Пуассона:

$$P_n(k) = \frac{\lambda^k}{k!}e^{-\lambda}, \text{ где } \lambda = np = 1000 * 0,002 = 2$$

Имеем:

$$P_{1000}(3) = \frac{2^3}{3!}e^{-2} \approx \frac{8}{6} \cdot 0,13534 \approx 0,18$$

Геометрическое распределение. Пусть $X_1, X_2, ..., X_(n)$ – конечная последовательность независимых случайных величин с распределением Бернулли. Построим случайную величину $Y = min_{1 \le i \le n}\{X_i = 1\} - 1$, обозначающее количество неудач до первого успеха. Случайная величина Y имеет геометрическое распределение с вероятностью успеха p, причем $P(Y = n) = q^n \cdot p$

Пример. Пусть игральный кубик выбрасывается до выпадения первой шестерки. Тогда вероятность того, что потребуется не больше трех бросаний, равна

$$P(Y \le 2) = P(Y = 0) + P(Y = 1) + P(Y = 2) =$$

$$= \left(\frac{5}{6}\right)^0 \cdot \left(\frac{1}{6}\right) + \left(\frac{5}{6}\right)^1 \cdot \left(\frac{1}{6}\right) + \left(\frac{5}{6}\right)^2 \cdot \left(\frac{1}{6}\right) = \frac{1}{6} \cdot \left(1 + \frac{5}{6} + \frac{25}{36}\right) = \frac{1}{6} \cdot \frac{91}{36} \approx 0,42$$

Гипергеометрическое распределение. В партии из N изделий имеется M (M < N) доброкачественных и N - M дефектных изделий. Если случайным образом из всей партии выбрать контрольную партию из n изделий, то число доброкачественных изделий в этой партии будет случайной величиной, которую обозначим Y. Ее распределение имеет вид:

$$P(Y = k) = \frac{C_m^k C_{N-M}^{n-k}}{C_N^n}$$

Пример. Пусть в урне находится 5 белых и 45 черных шаров. Вы закрываете глаза и вытаскиваете 10 шаров. Какова вероятность вытянуть ровно 4 белых шара?

В наших обозначениях N = 50, M = 5, n = 10, k = 4. Искомая вероятность равна

$$P(Y = 4) = \frac{C_5^4 C_{45}^6}{C_{50}^{10}} \approx 0,003964583$$

Функция распределения случайной величины $F_\zeta(x) = P(\zeta < x)$ удовлетворяет трем свойствам:

– $F_\zeta(x)$ является неубывающей функцией;

– $\lim\limits_{x\to\infty} F_\zeta = 0,\ \lim\limits_{x\to\infty} F_\zeta = 1$

– $F_\zeta(x)$ является непрерывной справа.

Пусть случайная величина X распределена равномерно на отрезке $[a; b]$. Ее функция распределения имеет вид:

$$F(X) = \begin{cases} 0, x < a \\ \frac{x-a}{b-a}, a \le x \le b \\ 1, x > b \end{cases}$$

Цепь Маркова. Пусть $\{E_1, E_2, ..., E_k\}$ – множество состояний некоторой физической системы. В любой момент времени система может находиться в одном состоянии и меняет свое состояние только в моменты $t_1, t_2, ..., t_n, ...$ Для однородных цепей Маркова вероятность p_{ij} перехода системы из состояния E_i в состояние E_j за один шаг зависит только от того, из какого состояния в какое осуществлялся переход.

Вероятности перехода p_{ij} удобно располагать в виде матрицы. Обозначим ее

$$P = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1k} \\ p_{21} & p_{22} & \dots & p_{2k} \\ \dots & \dots & \dots & \dots \\ p_{k1} & p_{k2} & \dots & p_{kk} \end{bmatrix}$$

и будем называть ***матрицей перехода*** однородной цепи Маркова за один шаг. Матрица P обладает следующими свойствами:

$$\text{а)}\ 0 \le p_{ij} \le 1 \qquad \text{б)}\ \sum_{j=1}^{k} p_{ij} = 1 (i = 1, 2, ..., k),$$

т.е. сумма элементов каждой строки матрицы перехода равна единице. Квадратные матрицы, для которых выполняются условия а) и б), называются **стохастическими**.

Вектор $a = (a_1, a_2, ..., a_k)$, где $a_i = P(E_i)$ – вероятность появления состояния $E_i (i = 1, 2, ..., k)$ в начальном испытании, называется вектором начальных вероятностей.

Матрица перехода $P^{(n)}$ за n шагов находится как P^n. Если из состояния E_i система может перейти в состояние E_j с положительной вероятностью за конечное число шагов, то говорят, что E_j достижимо из E_i. Состояние E_i называется существенным, если для каждого состояния E_j, достижимого из E_i, E_i достижимо из E_j. В противном случае E_i называется несущественным состоянием.

1. Какая вероятность? [Вальядолид, 10056]. n людей подбрасывают некоторую вещь (например, кубик), которая имеет несколько исходов. Если у некоторого игрока случается некоторый наперед установленный выигрышный исход (например, выпала цифра 3), то он объявляется победителем и игра останавливается. Вещь подбрасывается последовательно игроками: сначала первым, потом вторым и так далее. Если у n - ого игрока выигрышный исход не выпал, подбрасывание снова совершается первым игроком, потом вторым и так далее по очереди. Необходимо установить вероятность выигрыша i - го игрока.

Вход. Первая строка содержит количество тестов s ($s \leq 1000$). Каждая следующая строка является отдельным тестом и содержит три числа: количество игроков n ($n \leq 1000$), вероятность наступления победного события p и номер игрока i ($i \leq n$), вероятность выигрыша которого следует подсчитать.

Выход. Для каждого теста вывести вероятность выигрыша i - го игрока. Результат выводить с четырьмя знаками после десятичной точки.

Пример входа	Пример выхода
2	0.5455
2 0.166666 1	0.4545
2 0.166666 2	

2. Купоны [Вальядолид, 10288]. Имеется n разнотипных купонов и бесконечное количество закрытых коробок. В каждой коробке лежит один купон некоторого типа. Из каждой коробки с равной вероятностью можно извлечь купон любого типа. Какое ожидаемое количество коробок необходимо открыть, чтобы иметь хотя бы по одному купону каждого типа?

Вход. Каждая строка содержит число n ($1 \leq n \leq 33$), количество типов купонов.

Выход. Для каждого значения n вывести ожидаемое число коробок, которое надо открыть, чтобы иметь купоны всех типов. Если искомое число коробок целое, то вывести его. Если результат не целый, то вывести его целую часть, пробел, и дробную часть как показано в примере. Дробную часть результата представлять несократимой дробью.

Пример входа	Пример выхода
2	3
5 5	5 11 -- 12 5 11

3. Коровы и машины [Вальядолид, 10491]. Рассмотрим следующую телеигру. Имеется три двери, за двумя из которых спрятано по корове, а за третьими – приз. Игрок выбирает дверь, стараясь угадать, где находится приз. Ведущий после выбора игроком двери предлагает следующую сделку: он согласен открыть некоторую дверь, за которой находится корова в обмен на то, что игрок сменит дверь (поменяет свое мнение). Известно, что при смене мнения игрока вероятность выиграть приз составляет 2/3. В задаче обобщается эта игра. На вход подаются количество коров *ncows*, призов *ncars* и дверей *nshow*, открывающихся ведущим при смене мнения игрока. Найти вероятность выигрыша игрока, если он всегда меняет свое мнение. Известно, что за каждой дверью спрятана или корова, или приз.

Вход. Состоит из нескольких тестов. Каждый тест содержит три целых числа *ncows, ncars, nshow* $(1 \leq cows, cars \leq 10000, 0 \leq shown < cows)$.

Выход. Для каждого теста вывести в отдельной строке вероятность выигрыша игрока, если после предложения ведущего он сменит свое мнение. Ответ выводить с 5 десятичными знаками.

Пример входа	Пример выхода
2 1 1	0.66667
5 3 2	0.52500
2000 2700 900	0.71056

4. Подбрасывание кубиков [Вальядолид, 10759]. Подбрасывается n кубиков. Вычислить вероятность того, что сумма очков на всех кубиках будет как минимум x.

Вход. Каждая строка является отдельным тестом и содержит два целых числа: $n(1 \leq n \leq 24)$ и $x(0 \leq x < 150)$. Последний тест содержит $n = x = 0$ и не обрабатывается.

Выход. Для каждого теста вывести вероятность того, что сумма очков на всех кубиках будет как минимум x в формате, приведенном в примере. Все числа в ответе помещаются в беззнаковый 64 - битовый тип.

Пример входа	Пример выхода
3 9	20/27
1 7	0
24 24	1
15 76	11703055/78364164096
24 56	7895326546926586 45/7897302230536028 16
24 143	25/47383813388321616896
23 81	$\frac{1}{2}$
7 38	55/46656
0 0	

5. Боже! Спаси меня [Вальядолид, 10777]. В комнате имеется n дверей. Если Вы откроете дверь с номером i, то либо через x_i часов попадете в безопасное место, либо через x_i часов снова вернетесь в эту же комнату. Вычислить ожидаемое время P (в часах), через которое можно выбраться из комнаты в безопасное место.

Вход. Первая строка содержит количество тестов. Каждый тест содержит следующие данные. Первая строка каждого теста содержит число дверей $n(1 < n < 100)$. Далее следуют n строк, каждая из которых содержит числа $x_i, 0 < |x_i| < 25$ (если x_i положительно, то оно обозначает время, за которое можно попасть в безопасное место; если отрицательно, то $|x_i|$ – это время, через которое Вы снова окажетесь в комнате) и p_i – вероятность открыть i - ую дверь. Сумма всех p_i равна 1.

Выход. Для каждого теста вывести его номер, двоеточие, пробел, а затем фразу “God! Save me”, если выбраться из комнаты невозможно или ожидаемое время P (с двумя точками после запятой), через которое можно выбраться из комнаты в безопасное место.

Пример входа	Пример выхода
2	Case 1: 10.15
	Case 2: 9.76
3	
2 0.33	
-3 0.33	
-5 0.34	
3	
2 0.34	
-3 0.33	
-5 0.33	
2	

6. Хотите стать 2^n - эром? [Вальядолид, 10900]. Игрок имеет вначале игры 1$, ему задают n вопросов. На каждом шаге игрок может остановиться и забрать деньги или ответить на вопрос. Если на вопрос дан неправильный ответ, то игра заканчивается и игрок уходит без денег. Иначе призовая сумма удваивается и игра продолжается. После получения ответа на n - ый вопрос игра заканчивается, и игрок получает всю выигранную сумму.

Известно, что вероятность правильного ответа на каждый вопрос равна p и равномерно распределена на отрезке $[t..1]$.

Вход. Каждая строка содержит два числа: целое $n(1 \leq n \leq 30)$ и действительное $t(0 \leq t \leq 1)$. Последний тест содержит $n = t = 0$ и не обрабатывается.

Выход. Для каждого теста вывести величину ожидаемого выигрыша, если игрок будет придерживаться оптимальной стратегии. Результат округлять до трех десятичных знаков.

Пример входа	Пример выхода
1 0.5	1.500
1 0.3	1.357
2 0.6	2.560
24 0.25	230.138
0 0	

7. Заданная вероятность [Вальядолид, 11181]. n друзей собрались за покупками в супермаркет. Вероятность купить что-либо составляет $p_1, p_2, ..., p_n$ соответственно для каждого друга. После посещения магазина оказалось, что в точности r друзей совершили покупки (остальные ничего не купили). Определить вероятность покупательской способности каждого друга при выполнении этого условия.

Вход. Содержит не более 50 тестов. Первая строка каждого теста содержит два числа $n(1 \leq n \leq 20)$ и $r(0 \leq r \leq n)$. Каждая из следующих n строк содержит вероятность покупки i - го друга $p_i(0.1 \leq pi \leq 1)$. Все вероятности содержат как минимум два знака после десятичной точки. Последний тест содержит $n = r = 0$ и не обрабатывается.

Выход. Для каждого теста вывести его номер, а также n строк. i - ая строка содержит вероятность покупательной способности i - го друга при условии, что в точности r друзей совершили покупки.

Пример входа	Пример выхода
3 2	Case 1:
0.10	0.413043
0.20	0.739130
0.30	0.847826
5 1	Case 2:
0.10	0.200000
0.10	0.200000
0.10	0.200000
0.10	0.200000
0.10	0.200000
0 0	

8. Тройной прыжок [Топкодер, SRM 417, Дивизион 2, Уровень 3]. Тройной прыжок совершается следующим образом. Прыгун разгоняется, добегает до определенной отметки и совершает три последовательных прыжка. Победителем является тот, чья суммарная длина прыжков наибольшая.

Вы прыгаете последним. Все Ваши соперники уже совершили прыжки, их результаты заданы в массиве opponents.

Первый свой прыжок Вы уже совершили, его длина равна *first*. Длина каждого из оставшихся прыжков может с равной вероятностью принимать любое значение из отрезка [*lower, upper*], и не обязательно быть целым. Вы хотите вычислить вероятность того, что займете i - ое место. Необходимо построить массив длины $n + 1$ (n равно числу соперников), i -ый элемент которого (индексация массива начинается с 0) содержит вероятность того, что Вы займете (i + 1) - ое место.

Класс: TripleJump

Метод: vector<double> getProbabilities(int lower, int upper, int first, vector<int> opponents)

Ограничения: $1 \leq lower \leq 1000$, $lower \leq upper \leq 1000$, $lower \leq first \leq upper$, opponents содержит от 1 до 50 элементов, $1 \leq opponents[i] \leq 3000$.

Вход. Целые значения *lower, upper, first* и массив длин прыжков opponents.

Выход. Массив, содержащий вероятности того, что Вы займете первое, второе, ..., последнее место.

Пример входа

lower	upper	first	opponents
1	2	3	$\{1, 2, 3, 4\}$
3	7	5	$\{9, 9, 19, 19, 19\}$
1	10	5	$\{1, 2, 3, 5, 10, 11, 12, 19\}$

Пример выхода

$\{0.5, 0.5, 0.0, 0.0, 0.0\}$
$\{0.0, 0.0, 0.0, 1.0, 0.0, 0.0\}$
$\{0.2222, 0.6235, 0.0556, 0.0432, 0.0556, 0.0, 0.0, 0.0, 0.0\}$

9. Стрельба из лазера [Топкодер, SRM 431, Дивизион 1, Уровень 1]. Лазерная пушка расположена в точке (0, 0) на плоскости. Целями являются вертикальные отрезки с координатами концов $(x[i], y1[i]) - (x[i], y2[i])$. Выбирается произвольный угол от $-\pi/2$ до $\pi/2$ и производится выстрел. Выстрел под углом $-\pi/2$ производится вертикально вниз, 0 – горизонтально вправо, $\pi/2$ – вертикально вверх. Выстрелом является бесконечный луч, исходящий из начала координат. Выстрел попадает в цель, если луч и отрезок цели имеют общую точку.

Вычислить ожидаемое количество целей, которое может быть поражено одним выстрелом. Попадание в цель не меняет движение луча.

Класс: LaserShooting

Метод: double numberOfHits(vector<int> x, vector<int> y1, vector<int> y2)

Ограничения: x, y1 и y2 содержат одинаковое количество элементов, все элементы массива x разные, $1 \le x[i] \le 1000, -1000 \le y1[i], y2[i] \le 1000$.

Вход. Массивы x, y1 и y2, описывающие положения целей.

Выход. Ожидаемое количество целей, которое может быть поражено одним выстрелом.

Пример входа

x	y1	y2
$\{1\}$	$\{-1\}$	$\{1\}$
$\{3, 4, 7, 1\}$	$\{1, 2, 3, 4\}$	$\{4, 3, 2, 1\}$
$\{134, 298, 151, 942\}$	$\{-753, -76, 19, 568\}$	$\{440, 689, -39, 672\}$

Пример выхода

```
0.5
0.4623163952488826
1.444210260641501
```

10. Тир [Топкодер, TCHS 15, Уровень 2]. Вы поспорили с другом, что он не попадет в мишень, сделав n выстрелов. Вероятность попадания друга в мишень равна *accuracy*. Найти максимальное количество выстрелов, при котором Вам есть смысл спорить. Вам есть смысл спорить, если вероятность попадания друга в цель строго меньше 50%.

Класс: ShootingGallery

Метод: int profitableBet(int accuracy)

Ограничения: $1 \leq accuracy \leq 100$.

Вход. Вероятность попадания друга в мишень *accuracy*.

Выход. Максимальное количество выстрелов, при котором Вам есть смысл спорить.

Пример входа

accuracy
40
20
50
1

Пример выхода

```
1
3
0
68
```

11. Произвольное тасование [Топкодер, TCHS 47, Уровень 3]. Произвольное тасование чисел массива A производится согласно следующего алгоритма:

n = длина массива A

for i=1 to n

сгенерировать произвольное число r между 1 и n включительно

поменять местами A[i] и A[r] Вычислить вероятность того, что массив чисел outputArray получен в результате выполнения операции произвольного тасования набора $\{1, 2, ..., n\}$. Здесь n равно количеству элементов массива outputArray.

Класс: RandomShuffle

Метод: double probability(vector<int> outputArray)

Ограничения: outputArray содержит перестановку чисел от 1 до n, где n равно количеству элементов массива outputArray, $n \leq 10$.

Вход. Массив outputArray, содержащий перестановку чисел от 1 до n.

Выход. Вероятность того, что массив чисел outputArray получен в результате выполнения операции произвольного тасования набора чисел $\{1, 2, ..., n\}$.

Пример входа

outputArray
$\{1\}$
$\{1, 2\}$
$\{4, 2, 5, 1, 3\}$
1

Пример выхода

```
1.0
0.5
0.0067200000000000001
```

12. Игра в лотерею [Топкодер, TCHS 57, Уровень 2]. Возвращаясь домой, вы прочитали объявление: “Выберите m разных чисел между 1 и n включительно. Мы выберем m разных чисел между 1 и n произвольным образом. Если хотя бы k чисел совпадут, то Вы выиграли”.

В задаче требуется вычислить вероятность Вашего выигрыша.

Класс: TwoLotteryGames

Метод: double getHigherChanceGame(int n, int m, int k)

Ограничения: $2 \leq n \leq 8, 1 \leq m \leq n - 1, 1 \leq k \leq m$.

Вход. Целые числа n, m и k.

Выход. Вероятность выигрыша игрока.

Пример входа

n	m	k
3	2	1
8	2	1
8	4	2

Пример выхода

```
1.0
0.4642857142857143
0.7571428571428571
```

Указания и решения.

1. Какая вероятность? [Вальядолид, 10056]. Вероятность того, что i - ый игрок на первом же своем броске выиграет, равна $p * (1-p)^{i-1}$. Вероятность того, что i - ый игрок выиграет при втором своем броске, равна $p * (1-p)^{i-1} * (1-p)^n$: для этого необходимо чтобы первый бросок каждого игрока потерпел неудачу (вероятность $(1-p)^n$), далее первые $i-1$ игроков не получили выигрышный исход (вероятность $(1-p)^{i-1}$), и наконец, i - ый игрок выиграл, совершив свой бросок с вероятностью p.

Соответственно, i - ый игрок выиграет на k - ом своем броске с вероятностью $p * (1-p)^{i-1} * (1-p)^{kn}$

Просуммируем вероятности выигрыша i - ого игрока. Искомая вероятность его выигрыша равна

$$p * (1-p)^{i-1} + p * (1-p)^{i-1} * (1-p)^n + p * (1-p)^{i-1} * (1-p)^{2n} + ...+$$

$$+p * (1-p)^{i-1} * (1-p)^{kn} + ... =$$

$$= p * (1-p)^{i-1}(1 + (1-p)^n + (1-p)^{2n} + .. + (1-p)^{kn} + ...)$$

и образует бесконечную геометрическую прогрессию, сумма которой равна

$$p * (1-p)^{i-1} \frac{1}{1-(1-p)^n} = \frac{p * (1-p)^{i-1}}{1-(1-p)^n}$$

Отдельно следует обработать случай, когда $p = 0$. В таком случае ответом будет 0.

Реализация. Читаем входные данные. Если вероятность p равна 0, то выводим 0. Иначе вычисляем искомую вероятность при помощи приведенной выше формулы. Результат выводим с 4 знаками после десятичной точки.

```
#include <stdio.h>
#include <math.h>

int s,n,i;
```

```
double p,res;

int main(void)
{
  scanf("\%d",&s);
  while(s--)
  {
    scanf("\%d \%lf \%d",&n,&p,&i);
    if (p < 1e-7) printf("0.0000\n");
    else
    {
      res = p*pow(1-p,i-1)/(1-pow(1-p,n));
      printf("\%.4lf\n",res);
    }
  }
  return 0;
}
```

2. Купоны [Вальядолид, 10288]. Предположим, что у Вас уже имеется $n - k$ разных купонов. Обозначим через a_k ожидаемое количество коробок, которое следует открыть для того чтобы собрать недостающие k разных купонов. С вероятностью $(n - k)/n$ купон в следующей коробке будет бесполезным, а с вероятностью k/n он будет того типа, которого у Вас еще нет. Имеем соотношение:

$$a_k = (1 + ak) * \frac{n-k}{n} + (1 + a_{k-1}) * \frac{k}{n},$$
$$a_0 = 0$$

Раскроем скобки и выразим a_k через a_{k-1}:

$$a_k = \frac{n}{k} + a_{k-1}$$

Рекуррентность можно расписать в виде:

$$a_k = \frac{n}{k} + a_{k-1} = \frac{n}{k} + \frac{n}{k-1} + a_{k-2} = \frac{n}{k} + \frac{n}{k-1} + \frac{n}{k-2} + a_{k-3} = ... =$$

$$= \frac{n}{k} + \frac{n}{k-1} + \frac{n}{k-2} + ... + \frac{n}{1}a_0 = n * \sum_{i=1}^{k} \frac{1}{i}$$

Ответом задачи будет значение a_n – ожидаемое количество коробок, которое следует открыть для того чтобы собрать недостающие n разных купонов:

$$a_n = n * \sum_{i=1}^{k} \frac{1}{i}$$

Для вывода результата в требуемом формате остается реализовать суммирование при помощи рациональных чисел. Для сокращения дробей будем использовать функцию gcd, вычисляющую наибольший общий делитель.

Пример

$a_2 = 2 * (1 + \frac{1}{2}) = 3,$

$a_3 = 3 * (1 + \frac{1}{2} + \frac{1}{3}) = 3 + \frac{3}{2} + 1 = 5\frac{1}{2}.$

Реализация. В структуре RatNumber храним рациональное число: числитель x и знаменатель y.

```
struct RatNumber
{
  long long x,y;
} c, temp;
```

Функция gcd вычисляет наибольший общий делитель.

```
struct RatNumber
long long gcd(long long a, long long b)
{
  return (!b) ? a : gcd(b,a % b);
}
```

Сложение рациональных чисел a и b. Возвращаемая сумма является несократимой дробью.

```
struct RatNumber add(struct RatNumber a,struct RatNumber b)
{
  struct RatNumber res;
  res.x = a.x*b.y + a.y*b.x;
  res.y = a.y * b.y;
  d = gcd(res.x,res.y);
  if (d)
  {
    res.x  /= d;res.y  /= d;
```

```
  }
  return res;
}
```

Функция digits находит количество знаков числа x.

```
int digits(long long x)
{
  if (x < 10) return 1;
  return digits(x/10)+1;
}
```

Основной цикл программы. Читаем входное значение n.

```
  while(scanf("\%d",&n) == 1)
  {
    c.x = 0; c.y = 1; temp.x = 1;
```

Вычисление суммы $c = n * \sum_{i=1}^{n} \frac{1}{i}$

```
    for(i=1;i<=n;i++)
    {
      temp.y = i;
      c = add(c,temp);
    }
    c.x = n*c.x;
    d = gcd(c.x,c.y);
    c.x /= d; c.y /= d;
    d = c.x / c.y;
    c.x -= d*c.y;
```

Переменная d содержит целую часть результата c. Если знаменатель результата равен 1, то выводим только числитель. Иначе форматируем вывод ответа как показано в условии задачи.

```
    if (c.y > 1)
    {
      for(i=0;i<=digits(d);i++) printf(" ");
      printf("%lld\n",c.x);
      printf("%lld ",d);
      for(i=0;i<digits(c.y);i++) printf("-"); printf("\n");
      for(i=0;i<=digits(d);i++) printf(" ");
      printf("%lld\n",c.y);
    }
```

```
    else printf("%lld\n",d);
  }
```

3. Коровы и машины [Вальядолид, 10491]. Для решения задачи следует записать формулу условной вероятности.

Число дверей равно $doors = cows + cars$. Вероятность сначала угадать корову составляет $cows/doors$, машину - $cars/doors$. После смены мнения и открытия дверей с $shown$ коровами игрок может выбирать приз среди $doors - shown - 1$ закрытых дверей. Пусть вначале игрок указал на корову. Тогда среди закрытых дверей имеется $cars$ машин, и вероятность ее выигрыша составляет $cars/(doors - shown - 1)$

Если вначале игрок указал на машину, то среди закрытых дверей осталось $cars - 1$ машин и вероятность ее выигрыша составляет $(cars - 1)/(doors - shown - 1)$ Результирующая вероятность выиграть машину равна $(cows/doors) * (cars/(doors - shown - 1)) + (cars/doors) * ((cars - 1)/(doors - shown - 1)) = (cows * cars + cars * (cars - 1))/(doors * (doors - shown - 1))$.

Пример. Рассмотрим первый тест. В игре имеются 3 двери. Если сначала игрок укажет на дверь с призом (вероятность 1/3), то после смены мнения он проиграет. Если сначала будет выбрана дверь с коровой (вероятность 2/3), то после открытия двери с коровой и смены мнения игрок непременно попадет на дверь с призом и выиграет. Таким образом, вероятность выигрыша составит $1/3 * 0 + 2/3 * 1 = 2/3$.

Реализация. Читаем входные данные, вычисляем чисто дверей $doors$, находим ответ по выше приведенной формуле и выводим его. Поскольку входные данные целые, то чтобы избежать потери точности при делении, следует делимое перевести в действительный тип, умножив его на 1.0.

```
#include <stdio.h>
int cows,cars,shown,doors;
double res;
void main() {
  while(scanf("\%d \%d \%d",&cows,&cars,&shown) == 3) {
    doors = cows + cars;
    res = 1.0*((cars-1)*cars + cars*cows)/(doors*
    *(doors-shown-1));
    printf("\%.5lf\n",res);
  }
}
```

4. Подбрасывание кубиков [Вальядолид, 10759]. Учитывая ограничения на n и x, вычислим в ячейках массива res[i][j] вероятность того, что при бросании в точности i кубиков выпадет сумма j. Очевидно, что при бросании одного кубика вероятность выпадения любого значения от 1 до 6 одинакова и равна 1/6:

$$res[1][i] = \begin{cases} 1/6, 1 \le i \le 6 \\ 0, \text{иначе} \end{cases}$$

При бросании i кубиков может выпасть сумма j, если при бросании первых i - *1* кубиков выпадет сумма j - k, а при бросании i - ого кубика сумма k, *k = 1, 2, ..., 6*. Таким образом

$res[i][j] = res[i-1][j-6] * res[1][6] + res[i-1][j-5] * res[1][5] + ... + +res[i-1][j-1] * res[1][1] = \frac{1}{6} \sum_{k=1}^{6} res[i-1][j-k]$, учитывая что $res[1][6] = res[1][5] = ... = res[1][1] = 1/6$.

Вероятность того, что сумма очков при бросании n кубиков будет как минимум x, равна $s = \sum_{i=x}^{6 \cdot n} res[n][i]$

Пример. Рассмотрим пример заполнения массива res для одного, двух и трех кубиков. В пустых клетках вероятность равна 0.

i/j	1	2	3	4	5	6	7	8	9	10	11	12
1	1/6	1/6	1/6	1/6	1/6	1/6						
2		1/36	2/36	3/36	4/36	5/36	6/36	5/36	4/36	3/36	2/36	1/36
3			1/216	3/216	6/216	10/216	15/216	21/216	25/216	27/216	27/216	25/216

Обозначим через $P(n = i, s = j)$ вероятность того, что при бросании n кубиков выпадет сумма j. Рассмотрим вычисление значений некоторых ячеек.

а) На двух кубиках сумма 5 может получиться при следующих исходах: $1+4, 2+3, 3+2, 4+1$. То есть $P(n=2, s=5) = P(n=1, s=1) * P(n=1, s=4) + P(n=1, s=2) * P(n=1, s=3) + P(n=1, s=3) * P(n=1, s=2) + P(n=1, s=4) * P(n=1, s=1) = 1/6 * 1/6 + 1/6 * 1/6 + 1/6 * 1/6 + 1/6 * 1/6 = 4/36$

б) На трех кубиках может получиться сумма 10, если на первых двух выпало 4, на третьем - 6, или на первых двух 5, на третьем 5 и так далее.

При этом значения вероятностей для двух кубиков $P(n = 2, s = i)$ уже вычислены.

$P(n = 3, s = 10) = P(n = 2, s = 4) * P(n = 1, s = 6) +$
$+P(n = 2, s = 5) * P(n = 1, s = 5) + P(n = 2, s = 6) * P(n = 1, s = 4) +$
$+P(n = 2, s = 7) * P(n = 1, s = 3) + P(n = 2, s = 8) * P(n = 1, s = 2) +$
$+P(n = 2, s = 9) * P(n = 1, s = 1) = 3/36 * 1/6 + 4/36 * 1/6 + 5/36 * 1/6 +$
$+6/36 * 1/6 + 5/36 * 1/6 + 5/36 * 1/6 = 3/216 + 4/216 + 5/216 + 6/216 +$
$+5/216 + 4/216 = 27/216$

Реализация. Все вычисления будем проводить в 64 - битовом целочисленном типе long long. Переопределим его.

```
#define i64 long long
```

Поскольку вычисления следует производить с дробями, определим тип рационального числа в структуре RatNumber. Определим рабочий массив res и дополнительные переменные.

```
struct RatNumber {
  i64 x,y;
} one6,temp,res[25][150],s;
```

Для работы программы нам понадобятся функции вычисления наибольшего общего делителя (gcd) и наименьшего общего кратного (lcm).

```
i64 gcd(i64 a, i64 b)
{
  return (!b) ? a : gcd(b,a \% b);
}
i64 lcm(i64 a, i64 b)
{
  return a / gcd(a,b) * b;
}
```

Функция RatNumber складывает два рациональных числа по формуле

$$\frac{a.x}{a.y} + \frac{b.x}{b.y} = \frac{HOK(a.y, b.y)/a.y \cdot a.x + HOK(a.y, b.y)/b.y \cdot b.x}{HOK(a.y, b.y)}$$

Вычисляя наименьшее общее кратное знаменателей, можно избежать переполнения. Далее находим наибольший общий делитель числителя и знаменателя результата и сокращаем полученную сумму.

```
struct RatNumber add(struct RatNumber a,struct RatNumber b)
{
  struct RatNumber res;
  res.y = lcm(a.y,b.y);
  res.x = res.y/a.y*a.x + res.y/b.y*b.x;
  d = gcd(res.x,res.y);
  if (d)
  {
    res.x  /= d;res.y  /= d;
  }
  return res;
}
```

Основная часть программы. Установим значения рациональных чисел $res[i][j]$ равными 0, интерпретируя 0 как 0/1.

```
int main(void)
{
  for(i=0;i<25;i++)
  for(j=0;j<150;j++)
    res[i][j].x = 0, res[i][j].y = 1;
```

Определим рациональное число one6 = 1/6 и инициализируем им значения $res[1][i], i = 1, 2, ..., 6$.

```
  one6.x = 1; one6.y = 6;
  for(i=1;i<=6;i++) res[1][i] = one6;
```

Вычисляем значения $res[i][j]$ по выше приведенной рекуррентной формуле.

```
  for(n=2;n<25;n++)
  for(i=n-1;i<=6*(n-1);i++)
  for(j=1;j<=6;j++)
  {
    temp = res[n-1][i];
    temp.y = temp.y*6;
    res[n][i+j] = add(res[n][i+j],temp);
  }
```

Для каждой пары входных значений n и x находим вероятность того, что сумма очков на всех кубиках будет как минимум x. Для этого вычисляем сумму

$$s = \sum_{i=x}^{6 \cdot n} res[n][i]$$

```
  while(scanf("\%d \%d",&n,&x),n+x)
  {
    s.x = 0, s.y = 1;
    for(i = x; i <= 6*n; i++)
      s = add(s,res[n][i]);
```

Выводим искомую вероятность в виде дроби. Если результат равен 0, то выводим один 0. Если вероятность равна 1, то выводим 1.

```
    if (s.x == 0) printf("0\n"); else
    if (s.y == 1) printf("\%lld\n",s.x); else
    printf("\%lld/\%lld\n",s.x,s.y);
  }
  return 0;
}
```

5. Боже! Спаси меня [Вальядолид, 10777]. Обозначим через P ожидаемое время, через которое можно выбраться из комнаты в безопас-ное место. Тогда оно равно

$$P = \sum_{i=1}^{n} p_i \cdot t_i,$$

где t_i – время, через которое можно попасть в безопасное место, если пойти в i - ую дверь. Очевидно, что $t_i = x_i$, если $x_i > 0$. Если $x_i < 0$, то для того чтобы выбраться, следует потратить время $-x_i$ чтобы снова оказаться в комнате, а потом время P, через которое можно выбраться. То есть в этом случае $t_i = -x_i + P$. Таким образом, получаем линейное уравнение относительно P. Построить и решить уравнение можно за время $O(n)$, где n – количество дверей.

Пример. Рассмотрим входные данные для первого теста. Составим по ним уравнение: $P = 0.33 * 2 + 0.33 * (3 + P) + 0.34 * (5 + P)$, откуда $0.33 * P = 3.35$ или $P = 10.15$.

Реализация. Линейное уравнение будем решать следующим образом. Все слагаемые, в которых встречается множитель P, будем переносить влево, а коэффициент при нем хранить в переменной left. Все слагаемые-константы будем собирать справа, и хранить в переменной right. Изначально left = 1, right = 0. Читаем n пар чисел x и p. Если x > 0, то увеличиваем

правую сторону на величину x * p. Если x < 0, то в правой части уравнения появится слагаемое $p*(-x+P)$. И тогда следует прибавить к правой части величину $p*(-x)$, а из левой вычесть p.

```
#include <stdio.h>
void main(void)
{
  int i,j,n,tests;
  double left,right,x,p,res;
  scanf("\%d",&tests);
  for(i=1;i<=tests;i++)
  {
    left = 1.0; right = 0.0;
    scanf("\%d",&n);
    for(j=0;j<n;j++)
    {
      scanf("\%lf \%lf",&x, &p);
      if (x < 0.0)
      {
        left -= p;
        right += p*(-x);
      } else right += x*p;
    }
```

Имеем уравнение $left * P = right$. Если $left = 0$, то выбраться из комнаты невозможно (не существует двери, ведущей в безопасное место). Иначе искомое время равно $P = right/left$.

```
    if (left > 0.0)
    {
      res = right / left;
      printf("Case \%d: \%.2lf\n",i,res);
    } else printf("Case \%d: God! Save me\n",i);
  }
}
```

6. Хотите стать 2^n - эром? [Вальядолид, 10900]. Пусть $f(n,a)$ – максимально возможное значение выигрыша, если игроку будет задано n вопросов, a начальная сумма равна a. Если $n = 0$, то игрок остается с начальной суммой, то есть $f(0,a) = a$. Вероятность правильного ответа равна $p, t \leq p \leq 1$. Если на первый вопрос игрок отвечает правильно, то дальше ему остается ответить на $n-1$ вопросов имея призовую сумму $2a$. С вероятностью $1-p$ дается неверный ответ, и деньги пропадают. То есть

ожидаемая сумма выигрыша после первого ответа станет равной $p * f(n-1, 2a) + (1-p) * 0 = p * f(n-1, 2a)$. Если это значение больше предыдущей суммы a, то стоит отвечать на вопрос, иначе следует остановить игру. Ожидаемый выигрыш после ответа на вопрос составит $max(a, p * f(n-1, 2a))$. Поскольку вероятность p равномерно распределена на отрезке $[t..1]$, то

$$f(n, a) = \frac{1}{1-t} \int_t^1 max(a, p \cdot f(n-1, 2a))dp$$

Если $t = 1$, то вероятность дать правильный ответ равна 1 и в таком случае следует отвечать на все n вопросов, получив при этом выигрыш 2^n.

Пример. Рассмотрим третий тест, $n = 2, t = 0.6$. Начальный капитал $a = 1$.

$$f(2, 1) = \frac{1}{0,4} \int_{0.6}^1 max(1, p \cdot f(1, 2))dp,$$

$$f(1, 2) = \frac{1}{0.4} \int_{0.6}^1 max(0, p \cdot f(0, 4))dp, f(0, 4) = 4$$

Вычислим значение $f(1, 2)$ через $f(0.4)$:

$$f(1, 2) = \frac{1}{0,4} \int_{0.6}^1 max(0, p \cdot f(0, 4))dp = 2.5 \int_{0.6}^1 max(0, 4p)dp =$$

$$= 2.5 \int_{0.6}^1 4pdp = 2.5 \cdot 2p^2 \Big|_{0.6}^1 = 5 * (1 - 0.36) = 5 * 0.64 = 3.2$$

Вычислим значение $f(2, 1)$ через $f(1, 2)$:

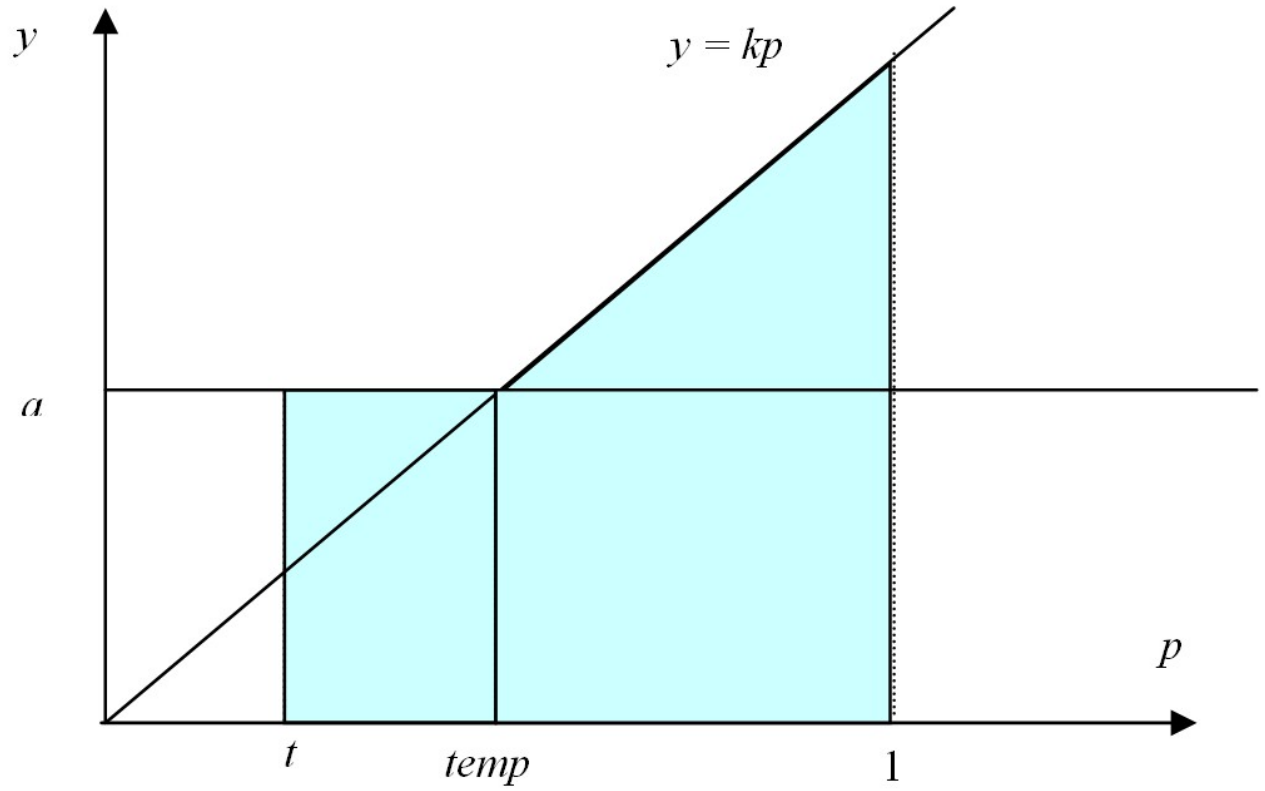
$$f(1, 2) = \frac{1}{0,4} \int_{0.6}^1 max(1, p \cdot f(1, 2))dp = 2.5 \int_{0.6}^1 max(1, 3.2p)dp =$$

/ учитываем, что $3.2p > 1$ при $0.6 \leq p \leq 1$ /

$$2.5 \int_{0.6}^1 3.2pdp = 2.5 \cdot 1.6p^2 \Big|_{0.6}^1 = 4 * (1 - 0.36) = 4 * 0.64 = 2.56$$

Реализация. Функция integral вычисляет значение интеграла $I(a, k) = \frac{1}{1-t} \int_t^1 max(a, kp)dp$ для заданных действительных чисел a и

k. При $t = 1$ вероятность угадывания p равна единице, значение интеграла $I(a, k)$ полагается равным $max(a, k)$. Ниже приведена область, площадь которой равна значению интеграла $\int_t^1 max(a, kp)dp$:



Найдем точку пересечения прямых $y = a$ и $y = kp$: $a = kp$, откуда $p = a/k$. Положим $temp = a/k$. Значение интеграла $\int_{0.6}^1 max(a, kp)dp$ рассмотрим как сумму $\int_t^{temp} adp + \int_{temp}^1 kpdp$. В зависимости от положения точки $temp$ относительно точек t и 1 вычисляем значение интеграла $I(a, k)$.

```
#include <stdio.h>
int n;
double t;
double max(double i, double j)
{
  return (i > j) ? i : j;
}
```

```
double integral(double a, double k)
{
  double s=0,temp = a/k;
  if (t == 1) return max(a,k);
  if (temp > 1) return a*(1-t);
  if (temp >= t) s = a*(temp - t);
  else temp = t;
  s += k*(1-temp*temp)/2;
  return s / (1 - t);
}
```

Рекурсивное вычисление $f(n,a) = \begin{cases} I(a, f(n-1,2a)), n > 0 \\ a, n = 0 \end{cases}$

```
double f(int n, int a)
{
  double k;
  if (!n) return a;
  k = f(n-1,2*a);
  return integral(a,k);
}
```

В основном цикле программы читаем входные данные и выводим значение $f(n,1)$.

```
void main(void)
{
  while(scanf("%d %lf",&n,&t),n+t>0)
    printf("%.3lf\n",f(n,1));
}
```

7. Заданная вероятность [Вальядолид, 11181]. Вычислим вероятность P того, что в точности *r* друзей совершат покупки. Для этого переберем все подмножества из *r* друзей (их не более C_{20}^{10}). Одновременно для каждого друга вычисляем вероятность pr_i того, что он совершит покупку при условии, что в точности *r* друзей купили что-нибудь. Таким образом, вероятность покупательской способности *i* - ого друга при условии совершении покупок в точности *r* друзьями, составит pr_i/P.

Пример. Рассмотрим первый тест. Двум купившим друзьям из трех будут соответствовать последовательности 011, 101, 110. Требуемые вычисления приведены в таблице:

последов.	подмножество	вероятность	prob[1]	prob[2]	prob[3]
011	{2, 3}	0.9 * 0.2 * 0.3 = 0.054	-	0.054	0.054
101	{1, 3}	0.1 * 0.8 * 0.3 = 0.024	0.024	-	0.024
110	{1, 2}	0.1 * 0.2 * 0.7 = 0.014	0.014	0.014	-
сумма вероятностей *all*		0.092	0.038	0.068	0.078
prob[*i*] / *all*			0.413043	0.739130	0.847826

Реализация. Исходные вероятности покупки друзей p_i храним в массиве p. В переменной *all* вычисляем вероятность того, что в точности r друзей из n совершат покупки. Вероятность покупки каждого друга при условии, что в точности r друзей купили что-нибудь, будем хранить в ячейках массива prob.

```
double p[21],prob[21],all;
int m[21];
```

Функция generate генерирует последовательности длины n, состоящие из $n-r$ нулей и r единиц. Каждой такой последовательности соответствует подмножество из r друзей. Для каждого подмножества друзей вычисляем вероятность их покупок и добавляем к *all*. Если в это подмножество входит i - ый друг, то вероятность прибавим и к prob[i].

```
void generate(int pos, int r)
{
  double p1;
  int i;
  if ((pos + r > n) || (r < 0)) return;
  if (!r && (pos == n))
  {
     for(p1=1.0,i=0;i<n;i++)
       if (m[i]) p1 *= p[i]; else p1 *= (1 - p[i]);
     all += p1;
     for(i=0;i<n;i++)
       if (m[i]) prob[i] += p1;
     return;
  }
  generate(pos+1,r);
  m[pos] = 1;
  generate(pos+1,r-1);
```

```
  m[pos] = 0;
}
```

Основной цикл программы. Читаем входные данные. Заданные вероятности сохраняем в массиве p.

```
while(scanf("%d %d",&n,&r),n+r)
{
  for(all=i=0;i<n;i++) scanf("%lf",&p[i]);
  memset(m,0,sizeof(m)); memset(prob,0,sizeof(prob));
```

Генерируем последовательности длины *n* из *r* единиц.

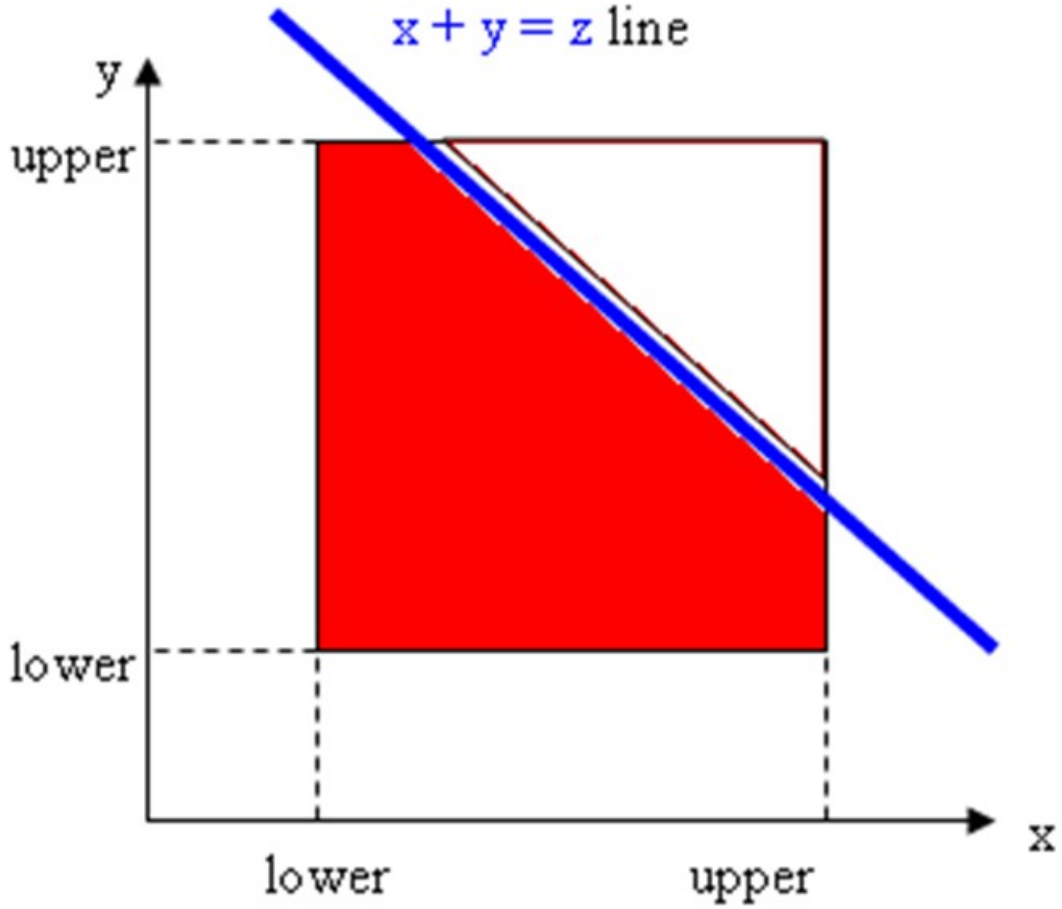
```
  generate(0,r); // pos, r
```

Для каждого друга выводим искомую вероятность.

```
  printf("Case %d:\n",cs++);
  for(i=0;i<n;i++)
    printf("%.6lf\n",prob[i]/all);
}
```

8. Тройной прыжок [Топкодер, SRM 417, Дивизион 2, Уровень 3]. Отнимем величину первого прыжка от всех результатов прыжков соперников, таким образом сведя задачу к “двойному” прыжку. Из условия задачи следует, что длина каждого прыжка является независимой случайной величиной, равномерно распределенной на отрезке $[lower, upper]$. Сумма двух прыжков также является случайной величиной, определенной на отрезке $[2 * lower, 2 * upper]$. Но она уже не будет равномерно распределенной.

Определим случайную величину F(z) = сумма двух прыжков не более z. F(z) пропорционально площади области квадрата под прямой $z = x + y$, изображенной красным цветом.



Функция распределения F(z)имеет вид:

$$F(z)=\begin{cases}0, z \le 2\cdot lower \\ \dfrac{\left(z-2\cdot lower\right)^2}{2\cdot\left(upper-lower\right)^2}, 2\cdot lower < z \le upper + lower \\ 1-\dfrac{\left(z-2\cdot upper\right)^2}{2\cdot\left(upper-lower\right)^2}, upper + lower < z \le 2\cdot upper \\ 1, z > 2\cdot upper\end{cases}$$

Отсортируем длины прыжков соперников по убыванию. Вероятность занять первое место равна 1 - F(opponents[0]). Вероятность занять второе место равна F(opponents[0]) - F(opponents[1]) и так далее. Вероятность

занять последнее место равна $F(opponents[n-1])$, где n - количество соперников.

```
#include <cstdio>
#include <vector>
#include <set>
#include <algorithm>
using namespace std;
double f(double lower, double upper, double z)
{
  if (z <= 2*lower) return 0.0;
  if (z <= lower + upper)
    return (z-2*lower)*(z-2*lower)/2/(upper-lower)/
    /(upper-lower);
  if (z <= 2*upper)
    return 1-(z-2*upper)*(z-2*upper)/2/(upper-lower)/
    /(upper-lower);
  return 1.0;
}
class TripleJump
{
public:
  vector<double> getProbabilities(int lower, int upper,
  int first,
                                   vector<int> opponents)
  {
     int i, n = (int)opponents.size();
     vector<double> res(n + 1, 0);
     for(i = 0; i < n; i++) opponents[i] -= first;
     sort(opponents.begin(),opponents.end(),
     greater<int>());
     res[0] = 1 - f(lower,upper,opponents[0]);
     res[n] = f(lower,upper,opponents[n-1]);
     for(i = 1; i < n; i++)
       res[i] = f(lower,upper,opponents[i-1]) -
       f(lower,upper,opponents[i]);
     return res;
  }
};
```

9. Стрельба из лазера [Топкодер, SRM 431, Дивизион 1, Уровень 1]. Математическое ожидание суммы случайных величин равно сумме мате-

матических ожиданий этих величин. Поэтому достаточно вычислить для каждой цели ожидаемое количество выстрелов для ее поражения и просуммировать полученные значения. Обозначим через a_1 и a_2 углы, при которых поражаются концы i - ой цели. Поскольку значения элементов массива x положительны, то $a_1 = arctg(y1[i]/x[i])$, $a_2 = arctg(y2[i]/x[i])$. i - ая цель поражается, если угол выстрела лежит между $min(a_1, a_2)$ и $max(a_1, a_2)$. Вероятность попадания в i - ую цель равна вероятности того, что точка, выбранная из отрезка $[-\pi/2, \pi/2]$, попадет в отрезок $[min(a_1, a_2)$ и $max(a_1, a_2)]$. Она равна $(max(a_1, a_2) - min(a_1, a_2))/\pi$.

```
#include <cstdio>
#include <vector>
#include <cmath>
#define PI acos(-1.0)
using namespace std;
class LaserShooting
{
public:
  double numberOfHits(vector<int> x, vector<int> y1,
  vector<int> y2)
  {
    double a1, a2, res = 0.0;
    int i;
    for(i = 0; i < x.size(); i++)
    {
      a1 = atan(1.0*y1[i]/x[i]);
      a2 = atan(1.0*y2[i]/x[i]);
      res += fabs(a1 - a2) / PI;
    }
    return res;
  }
};
```

10. Тир [Топкодер, TCHS 15, Уровень 2]. Вероятность того, что друг не попадет в мишень с одного выстрела, равна $1 - accuracy/100.0$. Вероятность того, что друг не попадет в мишень сделав n выстрелов, равна

$$(1 - accuracy/100.0)^n$$

Ответом задачи будет такое наименьшее n, для которого указанная выше вероятность станет не больше 0.5.

```
#include <stdio.h>
```

```
class ShootingGallery{
public:
  int profitableBet(int accuracy)
  {
    double miss = 1 - accuracy / 100.0,prob = miss;
    int n = 0;
    while (prob > 0.5) n++,prob *= miss;
    return n;
  }
};
```

11. Произвольное тасование [Топкодер, TCHS 47, Уровень 3]. Занесем в массив cur последовательность $\{1, 2, ..., n\}$. Будем моделировать все возможные процессы тасования, выбирая в качестве значения *r* все значения от 1 до *n*. Процесс тасования состоит из *n* итераций, на каждой из которых в качестве *r* можно выбрать любое из *n* чисел (от 1 до *n*). Таким образом, из последовательности $\{1, 2, ..., n\}$ в результате тасования можно получить nn других последовательностей, некоторые из которых возможно будут одинаковыми ($n^n > n!$). В переменной s подсчитаем, сколько раз в процессе моделирования из $\{1, 2, ..., n\}$ получится последовательность, заданная в массиве outputArray. Разделив найденное число *s* на n^n, получим искомую вероятность.

Функция shuffle имеет единственный параметр *pos* - номер проводимой итерации. Для прохождения по времени следует совершить следующее отсечение. Пусть производится *pos* итерация, а текущий массив cur отличается от исходного m в *c* позициях. Тогда если $c > 2 * (n - pos)$, то очевидно, что за оставшиеся $n - pos$ обменов невозможно из cur получить m. Это следует из того, что за каждый из оставшихся обменов мы можем переставлять максимум 2 элемента.

```
#include <cstdio>
#include <vector>
#include <cmath>
using namespace std;
int m[10],cur[10];
int n,s = 0;
void shuffle(int pos){
  int i,c;
  if (pos == n)
  {
    for(i=0;i<n;i++)
      if (m[i] != cur[i]) return;
```

```
      s++;
      return;
    }
    for(c=i=0;i<n;i++)
      if (m[i] != cur[i]) c++;
    if (c > 2*(n-pos)) return;
    for(i=0;i<n;i++)
    {
      swap(cur[i],cur[pos]);
      shuffle(pos+1);
      swap(cur[i],cur[pos]);
    }}
class RandomShuffle
{
public:
  double probability(vector<int> outputArray)
  {
    int i;
    n = outputArray.size();
    for(i=0;i<n;i++) m[i] = outputArray[i],cur[i] = i+1;
    shuffle(0);
    return 1.0*s/pow((double)n,(double)n);
  }
};
```

12. Игра в лотерею [Топкодер, TCHS 57, Уровень 2]. Рассмотрим случай, когда надо угадать 5 номеров из 39. Очевидно, что вероятность выигрыша составит $1/C_{39}^5$. Вычислим вероятность того, что угадано будет в точности 4 номера. В этом случае из 5 номеров, задуманных игроком, 4 номера должны находиться среди 5 выпавших (C_5^4 вариантов), а один номер должен находиться среди 39 - 5 = 34 не выпавших (C_{34}^1 вариантов).

В общем случае, для угадывания в точности x номеров, необходимо чтобы эти x номеров находились среди m выпавших, а остальные загаданные $m - x$ номеров игрока находились среди $n - m$ невыпавших. Вероятность угадывания в точности x номеров равна

$$\frac{C_m^x \cdot C_{n-m}^{m-x}}{C_n^m}$$

Для нахождения искомой вероятности следует просуммировать вероятности того, что будет угадано в точности x номеров для x от k до m

включительно. Функция $Cnk(k,n)$ вычисляет значение C_n^k.

```
#include <stdio.h>
int Cnk(int k, int n)
{
  long long res = 1;
  if (k > n) return 0;
  if (k > n - k) k = n - k;
  for(int i=1;i<=k;i++)
    res = res * (n-i+1) / i;
  return (int)res;
}
class TwoLotteryGames{
public:
  double getHigherChanceGame(int n, int m, int k)
  {
    double res=0;
    int x;
    for(x=k;x<=m;x++)
      res += Cnk(x,m)*Cnk(m-x,n-m);
    return res / Cnk(m,n);
  }};
```

Список использованных задач.

[Вальядолид] acm.uva.es/problemset: 10056 (Какая вероятность?), 10288 (Купоны), 10491 (Коровы и машины), 10759 (Подбрасывание кубиков), 10777 (Боже! Спаси меня), 10900 (Хотите стать 2n - эром?), 11181 (Заданная вероятность).

[Топкодер] www.topcoder.com/tc: SRM 417 (TripleJump), SRM 431 (LaserShooting), TCHS 15 (ShootingGallery), TCHS 47 (RandomShuffle), TCHS 57 (TwoLotteryGames).

Задачи и разборы

Задача A. Одинаковые шары

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Корабельная роща. 1898. При сравнении колорита "Корабельной рощи", позднего произведения Шишкина, с колоритом ранних его работ, ясно видно, что художник-пейзажист, подобно своим современникам-жанристам, перешел от локального понимания цвета к скупой, но целостной цветовой гамме. В основе этой гаммы лежит передача объединяющей изображение светотени.

В этой картине Шишкин нашел в цветовом объединении серовато-коричневых стволов елей и зелени мха на первом плане новое для себя тональное понимание цвета.

В непрозрачной закрытой урне с небольшим отверстием в крышке находится n шаров (n четно), ровно половина из которых черные, остальные – белые. У Вас есть симметрическая монета, которую Вы начинаете подбрасывать. Если выпадет орел, то Вы извлекаете белый шар, если решка - то черный. Когда в урне остается в точности два шара, то они оказываются одинакового цвета и смысла дальше бросать монету нет (до этого момента в урне обязательно присутствуют хотя бы один белый и хотя бы один черный шар). Какая вероятность того, что произойдет описанная ситуация?

Формат входного файла

Каждая строка является отдельным тестом и содержит количество шаров в урне n ($0 < n < 100000$, n четно).

Формат выходного файла

Для каждого теста в отдельной строке вывести вероятность того, что произойдет описанная в условии задачи ситуация. Вероятность выводить с 8 знаками после десятичной запятой.

Пример

a.in	a.out
6 10 256	0.62500000 0.72656250 0.94998552

Разбор задачи A. Одинаковые шары

Вычислим вероятность противоположного события X - того, что последними будут вынуты шары разного цвета. Для этого среди первых $n-2$ вынутых шаров ровно половина должна быть белыми, половина – черными.

Рассмотрим схему испытаний Бернулли, где вероятность успеха (например, появления белого шара) равна $p = 1/2$, а неуспеха $q = 1 - p = 1/2$. Вероятность того, что среди $n-2$ вынутых шаров будет ровно $(n-2)/2$ белых, равна

$$P(X) = C_{n-2}^{\frac{n}{2}-1} p^{\frac{n}{2}-1} q^{\frac{n}{2}-1} = \frac{(n-2)!}{\left(\left(\frac{n}{2}-1\right)!\right)^2}\left(\frac{1}{2}\right)^{n-2}$$

Остается для каждого входного n вычислить значение P(X) и вывести 1 - P(X). Хотя P(X) лежит в промежутке от 0 до 1 включительно, возможны проблемы с его непосредственным вычислением. Прологарифмируем равенство:

$$lnP(X) = ln(n-2)! - 2 \cdot ln(\frac{n}{2}-1)! - (n-2) \cdot ln2$$

После вычисления правой части равенства, возведем *e* в него, получив P(X). Логарифм факториала вычислим как сумму логарифмов:

$$lnn! = ln1 + ln2 + ln3 + ... + lnn$$

В связи с многократным входом, все ответы следует предвычислить и занести в массив.

Пример: Для $n = 6$ вероятность того, что последними будут вынуты шары разного цвета, равна

$$C_4^2 p^2 q^2 = 6 \cdot (\frac{1}{2})^4 = \frac{6}{16} = \frac{3}{8}$$

Вероятность того, что последними будут вынуты шары одного цвета, равна $1 - \frac{3}{8} = \frac{5}{8} = 0,625$.

Задача B. Дни рождения

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Рожь. 1878. *Пейзаж стал классическим и программным произведением, как для самого художника, так и для художественной критики его времени. На одном из эскизов к картине И.И. Шишкин сделал запись, которую можно считать его творческим кредо: "Раздолье, простор, угодье, рожь, Божья благодать, русское богатство". Лирическое отношение к отечественной природе, воспетое И.И. Шишкиным, было ярко выражено в стихах Н.А. Некрасова:*

Все рожь кругом, как степь живая,
Ни замков, ни морей, ни гор.
Спасибо, сторона родная,
За твой врачующий простор.

Известно, что в группе из 23 или более человек вероятность того, что хотя бы у двух из них дни рождения (число и месяц) совпадут, превышает 50%. Этот факт может показаться противоречащим здравому смыслу, так как вероятность одному родиться в определённый день года довольно мала, а вероятность того, что двое родились в конкретный день – ещё меньше, но является верным в соответствии с теорией вероятностей. Таким образом, факт не является парадоксом в строгом научном смысле – логического противоречия в нём нет, а парадокс заключается лишь в различиях между интуитивным восприятием ситуации человеком и результатами математического расчёта.

Для заданного количества людей вычислить вероятность того, что двое из них родились в один день года. Год считать равным 365 дням.

Формат входного файла

Каждая строка является отдельным тестом и содержит количество людей $n (1 < n < 400)$.

Формат выходного файла

Для каждого значения n в отдельной строке вывести вероятность того, что хотя бы у двух из n людей дни рождения (число и месяц) совпадают. Искомую вероятность выводить в процентах и округлять до 8 знаков после запятой как указано в примере.

Пример

b.in	b.out
2	0.27397260%
10	11.69481777%
23	50.72972343%
366	100.00000000%

Разбор задачи B. Парадокс дней рождения

Рассчитаем сначала, какова вероятность $p(n)$ того, что в группе из n человек дни рождения всех людей будут различными. Если $n > 365$, то в силу принципа Дирихле вероятность равна нулю. Если же $n \leq 365$, то будем рассуждать следующим образом. Возьмём наугад одного человека из группы и запомним его день рождения. Затем возьмём наугад второго человека, при этом вероятность того, что у него день рождения не совпадёт с днем рождения первого человека, равна $1-1/365$. Затем возьмём третьего человека, при этом вероятность того, что его день рождения не совпадёт с днями рождения первых двух, равна $1 - 2/365$.

Рассуждая по аналогии, мы дойдём до последнего человека, для которого вероятность несовпадения его дня рождения со всеми предыдущими будет равна $1 - (n-1)/365$. Перемножая все эти вероятности, получаем вероятность того, что все дни рождения в группе будут различными:

$$\overline{p(n)} = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{365}\right)$$

Тогда вероятность того, что хотя бы у двух человек из n дни рождения совпадут, равна

$$p(n) = 1 - \overline{p(n)}$$

Задача C. Толстая монета

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вид в окрестностях Дюссельдорфа. 1865. *Окончив в 1860 году Петербургскую Академию художеств с большой золотой медалью, Иван Шишкин получил право на поездку за границу. В 1861-1865 годах молодой художник странствовал по Европе, где посещал мастерские известных художников. По заказу Н.Д. Быкова он написал в 1865 году картину "Вид окрестностей Дюссельдорфа", за которую получил звание академика.*

В городе N люди настолько полюбили игру с монетой, что двух исходов (орел, решка) им оказалось мало. Поэтому решено было создать толстую монету, одним из исходов бросания которой было бы падение на ребро.

Какую наименьшую толщину должна иметь монета радиуса r, чтобы вероятность ее падения на ребро равнялась $1/n$? Считать, что монета имеет вид прямого кругового цилиндра, а поверхность, на которую она бросается, является клейкой (монета, коснувшись поверхности, падает на ребро или на одно из оснований).

Формат входного файла

Каждая строка является отдельным тестом и содержит целочисленный радиус монеты $r(0 < r < 100000)$ и целое значение $n(1 < n < 100)$.

Формат выходного файла

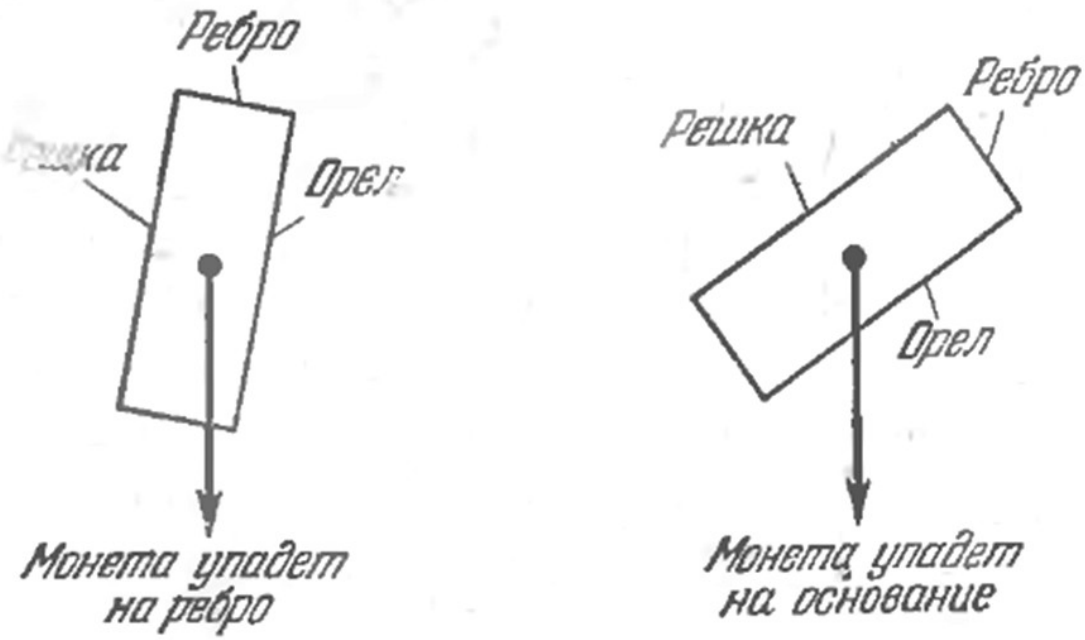
Для каждого теста в отдельной строке вывести искомую наименьшую толщину монеты. Результат выводить с точностью до 6 десятичных знаков.

Примеры

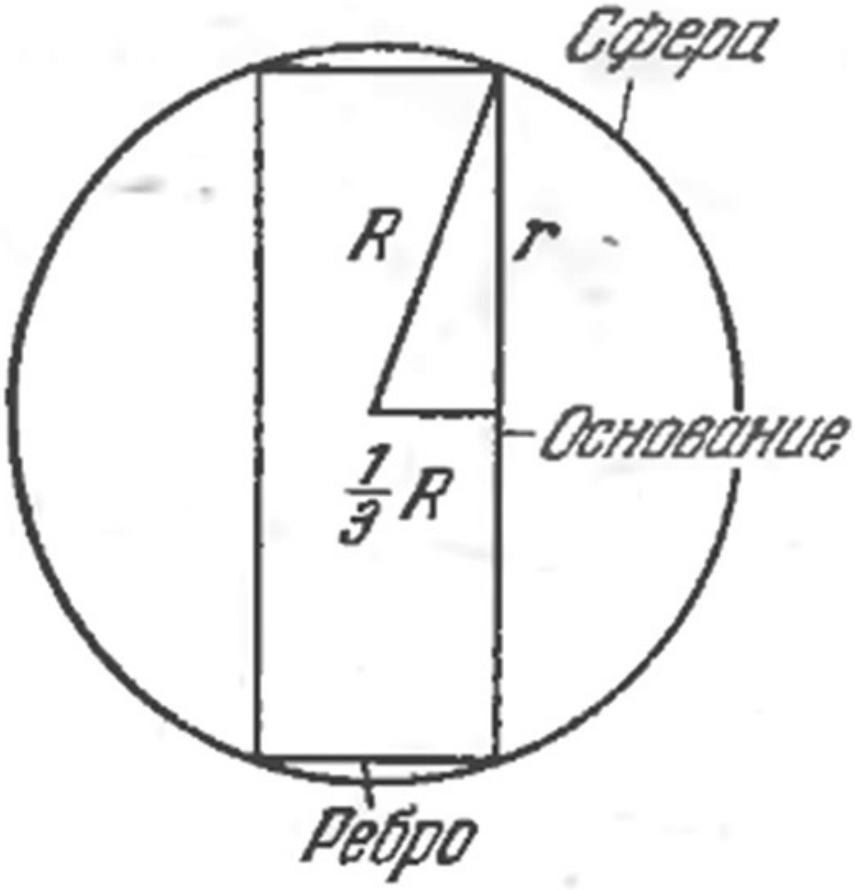
c.in	c.out
1 3 100 20 1000 50 0 0	0.707107 10.012523 40.008002

Разбор задачи C. Толстая монета

Рассмотрим монету как вписанную в сферу, центр которой совпадает с центром тяжести монеты. Считаем, что поверхность, на которую бросается монета, является клейкой



Лемма. Поверхность куска сферы, заключенного между двумя параллельными плоскостями, пропорциональна расстоянию между этими плоскостями. Поэтому толщина нашей монеты должна составлять $1/n$ диаметра сферы (на рисунке показан случай $n = 3$).



Пусть R – радиус сферы, а r – радиус монеты. По теореме Пифагора $R^2 = r^2 + \frac{1}{n^2}R^2$

или $\left(1 - \frac{1}{n^2}\right) R^2 = r^2,\ R^2 = \frac{r^2 n^2}{n^2 - 1}, R = \frac{rn}{\sqrt{n^2 - 1}}$

Откуда искомая толщина монеты должна быть как минимум

$2 * \frac{1}{n}R = \frac{r}{\sqrt{n^2 - 1}}$

Задача D. Дуэлянты

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

“Среди долины ровныя...”. 1883. *“Среди долины ровныя...” – классическое произведение Шишкина. Художник порой увлекался передачей поэтических мотивов, которые он видел в природе. Так, следуя некогда популярной песне А.Ф. Мерзлякова, в этой картине он пытался выразить величие могучего одинокого дуба, возвышавшегося среди широты полей.*

Среди долины ровныя,
На гладкой высоте,
Цветет, растет высокий дуб,
В могучей красоте.

Высокий дуб развесистый,
Один у всех в глазах;
Как рекрут на часах!
Взойдет ли красно солнышко -
Кого под сень принять?
Ударит ли погодушка -
Кто будет защищать?
Ах, скучно одинокому
И дереву расти!
Ах, горько, горько молодцу
Без милой жизнь вести!

Среди долины ровныя Петр Иванович и Илья Васильевич решили выяснить отношения на дуэли. Но никому из них не хотелось умирать. Поэтому они договорились, что каждый из них прибывает к дубу (место встречи) в случайный момент времени между 5 и 6 часами утра и, прождав соперника n минут, удаляется. В случае прибытия последнего в эти n минут дуэль состоится. Вычислить вероятность того, что дуэль закончится поединком.

Формат входного файла

Каждая строка является отдельным тестом и содержит целочисленное значение $n(0 < n \leq 60)$.

Формат выходного файла

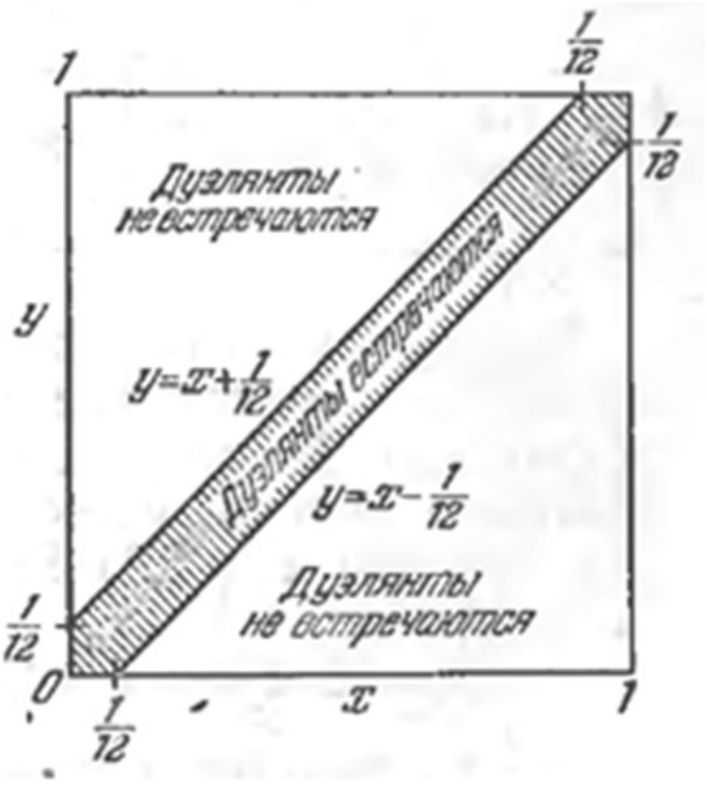
Для каждого теста в отдельной строке вывести вероятность того, что дуэль закончится поединком. Результат выводить с точностью до 6 десятичных знаков.

Примеры

d.in	d.out
3 5 8 10 5 8 10	0
2 6 7 5 8	1
5 1 2 3 4 5 2 3 4 5 1	4
5 1 2 3 2 1 0 0 9 0 0	6

Разбор задачи D. Распределение

Пусть x и y обозначают время прибытия первого и второго дуэлянтов соответственно, измеренное в долях часа, начиная с 5 часов. Заштрихованная площадь квадрата отвечает случаю, когда дуэлянты встречаются для случая $n = 5$ минут = 1/12 часа. Вероятность того, что дуэлянты не встретятся, равна $\left(1 - \frac{n}{60}\right)^2$ (сумма площадей двух треугольников).



Соответственно вероятность поединка составляет $1 - \left(1 - \frac{n}{60}\right)^2$.

Задача E. Лес

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Утро в сосновом лесу. 1889. Быть может, самая популярная картина Шишкина – "Утро в сосновом лесу". В ней он стремится передать очарование нетронутой девственной природы, поэтизируя жизнь леса.

Медведи написаны другом художника К.А. Савицким. Они получились столь удачно, что Савицкий даже расписался вместе с Шишкиным на картине. Однако П.М. Третьяков, купив картину, снял подпись. Он утвердил авторство одного Шишкина. Ведь в картине, говорил меценат, "начиная от замысла

и кончая исполнением, все говорит о манере живописи, о творческом методе, свойственных именно Шишкину".

Рассмотрим простую модель роста деревьев в сосновом лесу. Все деревья разделим на четыре группы: молодые саженцы (возрастная группа от 0 до 15 лет), молодые деревья (от 16 до 30 лет), средневозрастные деревья (от 31 до 45 лет) и старые (старше 45 лет). Длительность одного периода времени составляет 15 лет. В задаче необходимо определить количественное изменение популяции деревьев, если:

1. Определенный процент каждой группы деревьев погибает;

2. Выжившие деревья одной группы переходят в следующую возрастную группу. Старые деревья остаются старыми;

3. Умершие деревья заменяются молодыми саженцами. Таким образом, количество деревьев в лесу остается неизменным. Известно, что изначально лес состоит из n молодых саженцев. Известен процент деревьев, который погибает за один период времени в каждой возрастной группе: $p_{baby}, p_{young}, p_{middle-age}, p_{old}$. Необходимо определить среднее значение количества деревьев в каждой группе через $15k$ лет.

Формат входного файла

Каждая строка является отдельным тестом и содержит 6 целочисленных значений:

$n(10 < n <= 100000)$, $k(1 \leq k \leq 100)$,

$p_{baby}, p_{young}, p_{middle-age}, p_{old}.(0 \leq p_{baby}, p_{young}, p_{middle-age}, p_{old} \leq 100)$.

Формат выходного файла

Для каждого теста вывести в отдельной строке 4 значения – среднее число деревьев в каждой группе через $15k$ лет. Все числа выводить с 6 знаками после десятичной запятой.

Пример входа

```
50000 1 10 20 30 40
50000 2 10 20 30 40
50000 3 10 20 30 40
50000 100 10 20 30 40
```

Пример выхода

```
5000.000000 45000.000000 0.000000 0.000000
9500.000000 4500.000000 36000.000000 0.000000
12650.000000 8550.000000 3600.000000 25200.000000
12886.597938 11597.938144 9278.350515 16237.113402
```

Разбор задачи E. Лес

Обозначим через $b(k), y(k), m(k), o(k)$ количество деревьев соответственных возрастных групп через $15k$ лет. Значения p_{baby}, p_{young}, $p_{middle-age}$, p_{old} сразу поделим на 100, получив части умирающих деревьев в каждой группе.

Тогда согласно условию задачи, получим следующие соотношения:

$b(k+1) = p_{baby} * b(k) + p_{young} * y(k) + p_{middle-age} * m(k) + p_{old} * o(k)$

$y(k+1) = (1 - p_{baby}) * b(k)$

$m(k+1) = (1 - p_{young}) * y(k)$

$o(k+1) = (1 - p_{middle-age}) * m(k) + (1 - pold) * o(k)$

Обозначим вектор распределения деревьев через $x(k) = (b(k), y(k), m(k), o(k))$. Рассмотрим матрицу:

$$A = \begin{pmatrix} P_{baby} & P_{young} & P_{middle-age} & P_{old} \\ 1 - P_{baby} & 0 & 0 & 0 \\ 0 & 1 - P_{young} & & \\ 0 & 0 & 1 - P_{middle-age} & 1 - P_{old} \end{pmatrix}$$

Матрица A стохастическая, и $x(k+1) = A * x(k)$. Изначально $x(0) = (n, 0, 0, 0)$. Через $15k$ лет распределение деревьев составит $x(k) = Ak * x(0)$

Например, для $n = 50000$, $p_{baby} = 0,1$, $p_{young} = 0,2$, $p_{middle-age} = 0,3$, $p_{old} = 0,4$, получим

$$x(k) = 50000 \cdot \begin{pmatrix} 0,1 & 0,2 & 0,3 & 0,4 \\ 0,9 & 0 & 0 & 0 \\ 0 & 0,8 & 0 & 0 \\ 0 & 0 & 0,7 & 0,6 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$x(0) = \begin{pmatrix} 50000 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$x(1) = 50000 \cdot \begin{pmatrix} 0,1 & 0,2 & 0,3 & 0,4 \\ 0,9 & 0 & 0 & 0 \\ 0 & 0,8 & 0 & 0 \\ 0 & 0 & 0,7 & 0,6 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 50000 \cdot \begin{pmatrix} 1 \\ 0,9 \\ 0 \\ 0 \end{pmatrix}$$

$$x(2) = 50000 \cdot \begin{pmatrix} 0,1 & 0,2 & 0,3 & 0,4 \\ 0,9 & 0 & 0 & 0 \\ 0 & 0,8 & 0 & 0 \\ 0 & 0 & 0,7 & 0,6 \end{pmatrix} \cdot \begin{pmatrix} 0,1 \\ 0,9 \\ 0 \\ 0 \end{pmatrix} = 50000 \cdot \begin{pmatrix} 0,19 \\ 0,09 \\ 0,72 \\ 0 \end{pmatrix},$$

$$x(3) = 50000 \cdot \begin{pmatrix} 0,1 & 0,2 & 0,3 & 0,4 \\ 0,9 & 0 & 0 & 0 \\ 0 & 0,8 & 0 & 0 \\ 0 & 0 & 0,7 & 0,6 \end{pmatrix} \cdot \begin{pmatrix} 0,19 \\ 0,09 \\ 0,72 \\ 0 \end{pmatrix} = 50000 \cdot \begin{pmatrix} 0,253 \\ 0,171 \\ 0,072 \\ 0,504 \end{pmatrix}$$

Для матрицы $A = \begin{pmatrix} 0,1 & 0,2 & 0,3 & 0,4 \\ 0,9 & 0 & 0 & 0 \\ 0 & 0,8 & 0 & 0 \\ 0 & 0 & 0,7 & 0,6 \end{pmatrix}$ стабильным состоянием распределения деревьев будет $P = \dfrac{1}{3,88} \begin{pmatrix} 1 \\ 0,9 \\ 0,72 \\ 1,26 \end{pmatrix} = \begin{pmatrix} 0,257 \\ 0,232 \\ 0,186 \\ 0,325 \end{pmatrix}$

Задача F. Счастливые пятерки

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Сосновый бор. *Мачтовый лес в Вятской губернии. 1872. Крамской, друг художника, писал о "Сосновом боре", подчеркивая натурную достоверность изображенного: "Лес глухой и ручей с железистой, темно-желтой водой, в котором видно все дно, усеянное камнями..." Однако необходимо отметить, что Шишкин на своем полотне не-сколько театрализует лесной пейзаж, предлагая "природный спектакль".*

Некоторые люди считают пятерку счастливым числом. Они бросают несколько костей, и если пятерка выпадает больше чем в одной пятой части костей, то день считается счастливым. Бросается *dice* одинаковых костей, имеющих *sides* сторон. Необходимо вычислить вероятность того,

что день будет счастливым. Вероятность выпадения пятерки при бросании кости с *sides* сторонами равна $1/sides$.

Формат входного файла

Каждая строка является отдельным тестом и содержит количество бросаемых костей dice ($1 \leq dice \leq 20$) и количество сторон sides ($5 \leq sides \leq 10$) в них.

Формат выходного файла

Для каждого теста в отдельной строке вывести вероятность того, что день будет счастливым. Вероятность выводить с 6 знаками после десятичной запятой.

Пример

f.in	f.out
1 6 5 6 20 10 1 10	0.166667 0.196245 0.043174 0.100000

Разбор задачи F. Счасливые пятерки

Заведем двумерный массив p[21][21], в котором $p[d][i]$ равно вероятности выпадения в точности i пятерок при бросании d костей. Занесем в переменную q вероятность выпадения пятерки на кости, состоящей из *sides* сторон. Изначально обнулим массив p, присвоим p[0][0] = 1 (вероятность выпадения 0 пятерок при бросании 0 костей равна 1). При бросании d костей выпадет в точности i пятерок, если:

а) после бросания $d-1$ кости выпало i пятерок, а на d - ой кости выпала не пятерка. Вероятность этого события равна
$p[d-1][i] * (1-q)$.

б) после бросания $d-1$ кости выпала $i-1$ пятерка, а на d - ой кости выпала пятерка. Вероятность этого события равна $p[d-1][i-1] * q$.

Таким образом, $p[d][i] = p[d-1][i] * (1-q) + p[d-1][i-1] * q$

Пересчитаем по этой формуле все значения $p[d][i]$, $1 \leq d \leq dice$, $0 \leq i \leq dice$. День считается удачным, если пятерка выпадет как минимум на $dice/5+1$ кости (больше чем в одной пятой части). Для нахождения требуемой вероятности остается просуммировать значения $p[dice][i]$ для i от $dice/5+1$ до $dice$.

Задача G. Произведение матриц

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

В лесу графини Мордвиновой. Петергоф. 1891. *С годами у художника развилось обостренное цветовое видение. Теперь он видит и может передать кистью изменчивость цветов в зависимости от освещения и от рефлексов соседних предметов. Живописец выписывает мягкие переходы зеленых, желтоватых и сероватых оттенков стволов елей, хвои и мха. Но тонкие колористические изменения не самоцель для художника: ими он стремится донести до зрителя реальную жизнь природы. Картина вызывает у зрителя впечатление, что он находится внутри лесного пространства, дает прочувствовать окружающую атмосферу сосновой чащобы.* Даны три квадратные матрицы A, B, C, каждая из которых имеет размер $n \times n$. Необходимо проверить равенство: $A \times B = C$.

Формат входного файла

Каждый тест начинается значением $n (n \leq 500)$. Далее следуют три матрицы A, B, C, каждая из которых представляется n строками, содержащих в точности n чисел. Элементы матриц A и B по модулю не превышают 1000. Последний тест содержит $n = 0$ и не обрабатывается. Например, в первом тесте следует проверить равенство

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 & 3 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 5 & 9 \\ 11 & 21 \end{pmatrix}$$

Формат выходного файла

Для каждого теста в отдельной строке вывести “YES” или “NO” в зависимости от того, имеет ли место равенство $A \times B = C$ или нет.

Примеры

g.in	g.out
2 1 2 3 4 1 3 2 3 5 9 11 21	YES
2 1 2 3 4 1 3 2 3 5 9 10 21 0	NO

Разбор задачи G. Произведение матриц

Тесты были подобраны так, что простое умножение матриц с временной оценкой $O(n^3)$ дает Time Limit. Задачу следует решать вероятностным методом. Сгенерируем вектор r длины n из 0 и 1. Вычислим вектора $ABr = A(Br)$ и Cr за $O(n^2)$. Если они равны, то с вероятностью 50% можно сказать, что $A \times B = C$. Проведем такое тестирование, например, 10 раз. Тогда с вероятностью $1 - 1/2^{10}$ получим правильный ответ.

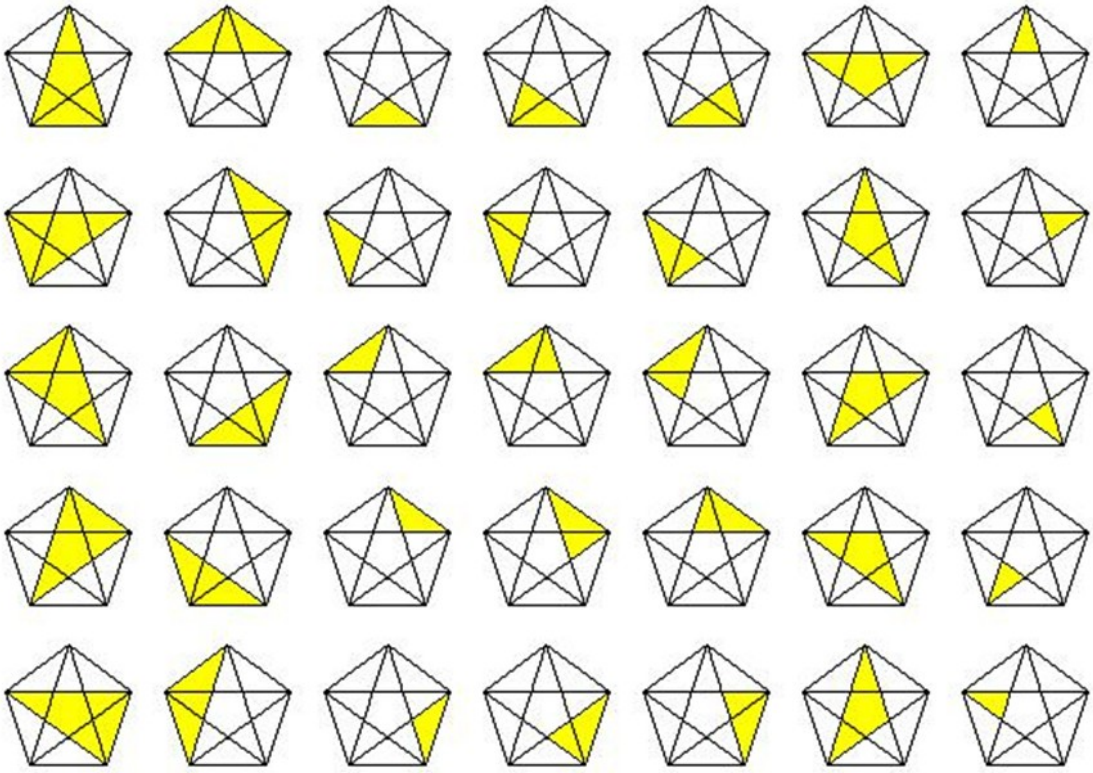
Задача H. Треугольники в многоугольнике

Имя входного файла:	h.in
Имя выходного файла:	h.out
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дождь в дубовом лесу. 1891. *Мотивы жанровой живописи, введен-ные в пейзаж, чрезвычайно редки для Шишкина, хотя именно этот прием во многом способствовал успеху картины у широкой публики. Но истинную ценность этого произведения составляет прекрасно выражен-*

ное состояние природы: начавшийся мелкий, моросящий дождь; мокрые, скользкие тропинки; серо-водянистый воздух.

В выпуклом n - угольнике провели все диагонали, никакие три из которых не проходят через одну точку. Сколько образовалось разных треугольников?



Формат входного файла

Каждая строка является отдельным тестом и содержит натуральное число $n(2 < n <= 1000)$.

Формат выходного файла

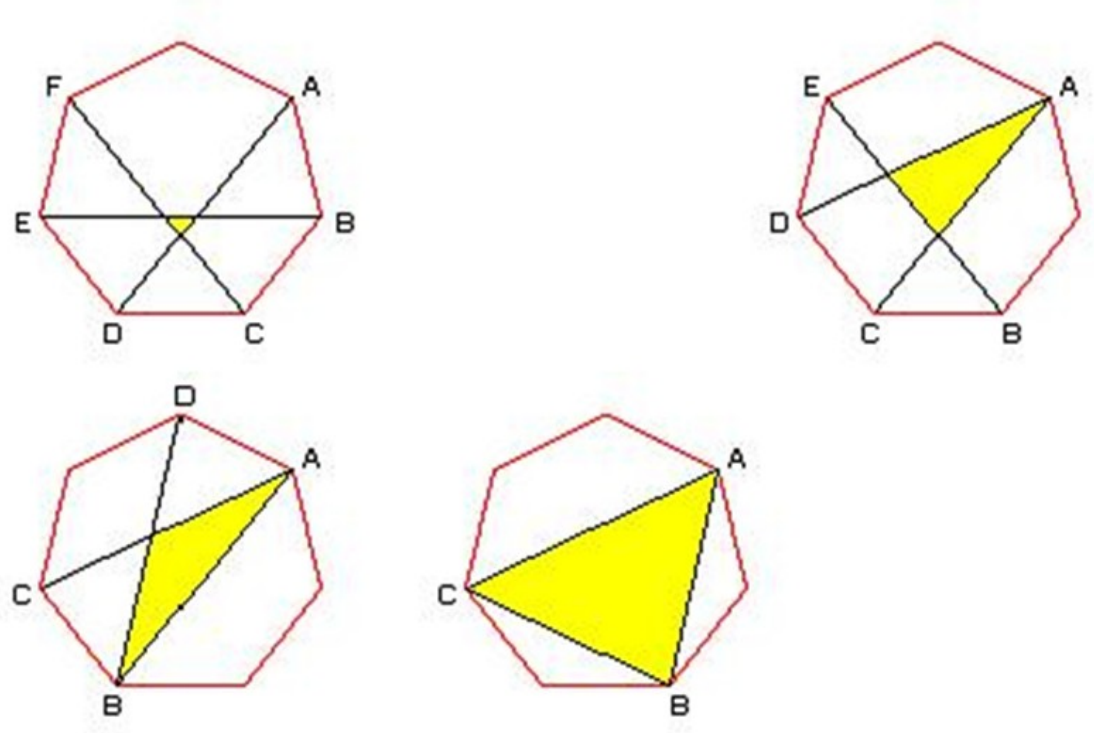
Для каждого теста в отдельной строке вывести количество разных треугольников, которое получится после проведения всех диагоналей.

Примеры

h.in	h.out
3 4 5	1 8 35

Разбор задачи H. Треугольники в многоугольнике

Каждый из образованных треугольников может иметь 0, 1, 2 или 3 общие вершины с многоугольником:



а) 0 общих вершин б) 1 общая вершина в) 2 общие вершины г) 3 общие вершины

Каждый треугольник, который не имеет общих вершин с многоугольником, образуется тремя диагоналями, которые в свою очередь опре-деляются 6 точками (а). Количество таких треугольников равно C_n^6. Треугольники, лишь одна вершина которых совпадает с вершиной многоугольника (б), образуются пятью точками, любая из которых может быть вершиной. Таких треугольников $5 * C_n^5$. Треугольники, две вершины которых лежат в вершинах многоугольника, определяются 4 точками (в). При этом каждые 4 точки образуют 4 разные треугольника. Их общее число равно $4 * C_n^4$. Количество треугольников, все вершины которых лежат в вершинах многоугольника (г), равно C_n^3. Таким образом, общее количество треугольников равно $f(n) = C_n^6 + 5 * C_n^5 + 4 * C_n^4 + C_n^3$

Например, количество треугольников в пятиугольнике со всеми проведенными диагоналями равно $C_5^6 + 5 * C_5^5 + 4 * C_5^4 + C_5^3 = 0 + 5 + 20 + 10 = 35$.

Задача I. Обработка текста

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Сестрорецкий бор. 1896. *В картине "Сестрорецкий бор" Шишкин остался верен своей манере: он достиг достоверности образа в сочетании с широким обобщением и типизацией изображения.*

Входной текст состоит из набора строк, каждая из которых имеет длину в точности 60 символов. Среди строк присутствуют следующие:

1. Vinnytsia National Technical University
2. Ivan Franko Lviv National University
3. 2009 Ukrainian Winter School takes place in Kharkiv
4. Taras Shevchenko Kiev National University

Эти строки, как и все остальные, имеют длину 60 символов (в конце каждой строки стоит недостающее количество пробелов). Известно, что количество этих строк в тексте составляет не более 10% всех строк. Необходимо определить относительное процентное присутствие каждой из указанных четырех строк относительно друг друга.

Формат входного файла

Текст в виде набора строк, длина каждой из которых составляет в точности 60 символов. Объем входного текста составляет 60 Мегабайт.

Формат выходного файла

Четыре натуральных числа, описывающих относительное процентное присутствие каждой из указанных строк относительно друг друга. Процентное содержание каждой строки следует округлять до ближайшего целого. Первое число указвает процентное содержание первой строки (Vinnytsia National Technical University) относительно всех четырех. Второе число указвает процентное содержание второй строки (Ivan Franko Lviv National University) относительно четырех и так далее.

Пример входа

```
50000 1 10 20 30 40
Taras Shevchenko Kiev National University
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345678
ABCD:'.HIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345678
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345678
ABCDEFGHdrthedrbhdrhrthbdrthcdefghijklmnopqrstuPwxyz12345678
ABCDEFGz|||LMNOPQRSTUVWXYZabцyefghijklmnopqrstuPwxyz12345678
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345678
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyasrgserg8
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345673
ABCDEFGHIJKLMNOPQRSTU%$#$#&%$#%&klmnopqrstuPwxyz12345678
ABCDEFGHIJKLMNOPQRSTUVWXYZabcde.,m.nm,.p;pjgyujPwxyz12345678
Ivan Franko Lviv National University
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345678
ABCDEFGHIJKLMfghjdgnhjrs4e5hwe45hhijklmnopqrstuPwxyz12345678
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345678
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345678
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgsdsgdfbsfthstuPwxyz12345678
adgsdfgsertsdfgPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345678
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuPwxyz12345678
whjkfghyjghjMNOsdfhgshsghsgfhsfghsfghlmnopqrstuPwxyz12sdfgsr
```

Пример выхода

```
0 50 0 50
```

Разбор задачи I. Обработка текста

Поскольку длины всех строк одинаковы, можно воспользоваться методом Монте-Карло и функцией fseek. Начало любой строки текста начинается с позиции, кратной 62 (60 символов строки + два символа перевода каретки OD OA). Объем текста составляет 60 Мегабайт, 10% которого занимают четыре указанные строки. Следовательно, всего в тексте около 1 миллиона строк, 100 тысяч из которых составляют четыре указанные строки.

Читаем 100000 произвольных строк, подсчитываем сколько раз каждая из четырех указанных строк среди них встретилась. Вычисляем требуемое процентное соотношение. Округляя проценты до целого значения, мы избегаем ошибок метода Монте-Карло.

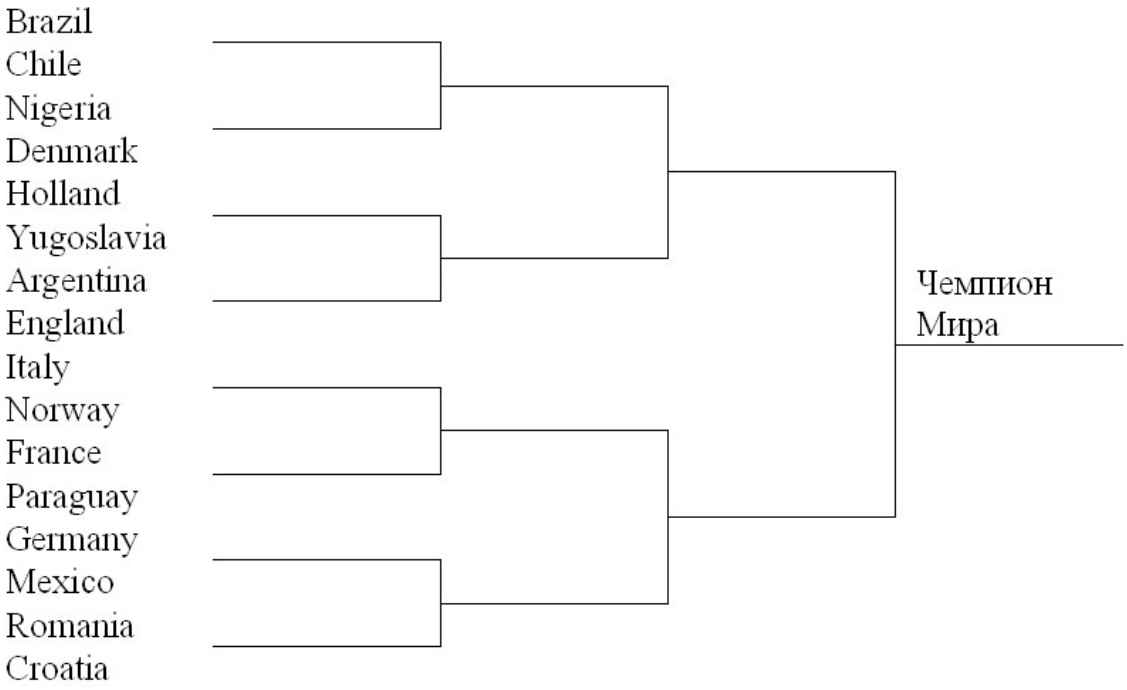
Задача J. Чемпионат мира

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Лес весной. 1884. *Пейзаж отличается тонкой градацией оттенков цвета, свободой и многообразием живописных приемов при сохранении строгого, реалистически точного рисунка.*

В финале чемпионата мира по футболу во Франции принимали участие 16 команд. Победитель определялся по олимпийской системе:

Известна вероятность выигрыша (в процентах) каждой команды над каждой. Необходимо для каждой команды вычислить вероятность того, что она выиграет турнир (станет чемпионом мира).



Формат входного файла

Состоит из нескольких тестов. Первая строка каждого теста содержит количество команд в чемпионате $n (4 \leq n \leq 64$, n является степенью двойки). Следующие n строк описывают названия команд, каждая из которых

содержит не более 10 символов. Далее следует матрица вероятностей p размера $n \times n$. Ячейка p[i][j] содержит неотрицательное целочисленное значение вероятности в процентах, с которой i - ая команда выиграет у j - ой. Очевидно, что p[i][j] + p[j][i] = 100%.

Формат выходного файла

Для каждого теста вывести его номер. Для каждой команды определить вероятность (в процентах) того, что она станет чемпионом мира. Названия команд выводить в том же порядке, в котором они подаются на вход. При выводе названия команд выравнивать слева, под каждую команду отводить 10 символов. После названия команды следует один пробел, после чего выводится процентная вероятность ее победы в турнире как показано ниже.

Пример входа

```
16
Brazil
Chile
Nigeria
Denmark
Holland
Yugoslavia
Argentina
England
Italy
Norway
France
Paraguay
Germany
Mexico
Romania
Croatia
50 65 50 60 55 50 50 65 45 55 40 55 40 55 50 50
35 50 35 45 40 35 35 50 30 40 25 40 25 40 35 35
50 65 50 60 55 50 50 65 45 55 40 55 40 55 50 50
40 55 40 50 45 40 40 55 35 45 30 45 30 45 40 40
45 60 45 55 50 45 45 60 40 50 35 50 35 50 45 45
50 65 50 60 55 50 50 65 45 55 40 55 40 55 50 50
50 65 50 60 55 50 50 65 45 55 40 55 40 55 50 50
35 50 35 45 40 35 35 50 30 40 25 40 25 40 35 35
55 70 55 65 60 55 55 70 50 60 45 60 45 60 55 55
45 60 45 55 50 45 45 60 40 50 35 50 35 50 45 45
```

```
60 75 60 70 65 60 60 75 55 65 50 65 50 65 60 60
45 60 45 55 50 45 45 60 40 50 35 50 35 50 45 45
60 75 60 70 65 60 60 75 55 65 50 65 50 65 60 60
45 60 45 55 50 45 45 60 40 50 35 50 35 50 45 45
50 65 50 60 55 50 50 65 45 55 40 55 40 55 50 50
50 65 50 60 55 50 50 65 45 55 40 55 40 55 50 50
```

Пример выхода

```
Brazil      p=8.54%
Chile       p=1.60%
Nigeria     p=8.06%
Denmark     p=2.79%
Holland     p=4.51%
Yugoslavia  p=7.50%
Argentina   p=8.38%
England     p=1.56%
Italy       p=9.05%
Norway      p=3.23%
France      p=13.72%
Paraguay    p=3.09%
Germany     p=13.79%
Mexico      p=3.11%
Romania     p=5.53%
Croatia     p=5.53%
```

Разбор задачи J. Чемпионат мира

Положим prob[i][j] = p[i][j] / 100, после чего значения вероятностей будут принимать значения от 0 до 1. В финале имеется 5 раундов (например, для n = 16): нулевой – начальный, с которого стартуют все команды (и вероятность в нем оказаться для каждой команды равна 1.0), четвертый – завершающий, в котором определяется чемпион мира. Заведем массив adv, в котором adv[i][j] будут содержать вероятность того, что j - ая команда пройдет в i - ый раунд. Будем последовательно пересчитывать ячейки массивов adv[1], adv[2], adv[3] и adv[4].

Заметим, что команда с номером j для прохода в i - ый раунд должна:

1. Пройти в (i - 1) - ый раунд;

2. Победить одну из команд, которая попадется ей на пути чемпионата согласно приведенной выше таблице. При этом j - ой команде в i - ом

раунде может попасться команда с одним из следующих номеров: j xor 2^{i-1}, j xor (2^{i-1} + 1), ..., j xor (2^i - 1).

Следовательно, вероятность adv[i][j] вычисляется по формуле:

adv[i][j] = adv[i - 1][j] * (adv[i - 1][j xor 2^{i-1}] * prob[j][j xor 2^{i-1}] +

+ adv[i - 1][j xor 2^{i-1} + 1] * prob[j][j xor 2^{i-1} + 1] + ... +

+ adv[i - 1][j xor 2^i - 1] * prob[j][j xor 2^i - 1])

Однако и такой вариант тоже возможен.

День шестой (23.02.2009 г.)
Контест Теодора Заркуа и его учеников

Об авторе...

1. **Николоз (Ника) Джимшелеишвили**, родился в 1987 году в городе Цхалтубо. Закончил физмат школу им. Размадзе в г.Кутаиси. Магистрант второго года Тбилисского ГУ им. И.Джавахишвили. Неоднократный призер IOI и IMO. Неоднократный чемпион Южного Кавказа среди студентов как в командном, так и в личном зачетах. Третий призер Открытых чемпионатов Москвы 2006, 2008 годов, третий призер Открытых чемпионатов Украины 2006, 2007 годов, Чемпион Украины 2008 года, неоднократный призер международных олимпиад им.Лебедева и Глушкова, Поттосина, Векуа. Двукратный финалист ТССО и финалист ТСО. Победитель Кубка Векуа 2008 в личном зачете. Обладатель дипломов разных степеней и разных лет Открытого Кубка и Командного чемпионата РФ по программированию среди студентов. Многократный чемпион Грузии по программированию среди студентов. Обладатель красного рейтинга сайта TopCoder. Имеет спортивный разряд по шахматам. Хобби - футбол. Холост.

2. **Эльдар Богданов**, родился в 1988 году в г.Тбилиси. Закончил физмат школу им.И.Векуа. Занимался программированием в учебном центре “Мзиури”. Студент четвертого курса ТбГУ им.И.Джавахишвили. Призер республиканской Олимпиады школьников по информатике 2005 года. Третий призер Открытого чемпионата Москвы 2008 года, Кубка Векуа 2008 и Открытого чемпионата Украины 2007 года, Второй призер Открытого чемпионата Южного Кавказа 2007 года и Олимпиады им.Лебедева и Глушкова 2008 года. Чемпион Грузии, Южного Кавказа и Украины 2008 года. Обладатель дипломов Открытого Кубка и Командного чемпионата РФ по программированию среди студентов.Обладатель желтого рейтинга сайта TopCoder. Хобби - музыка. Холост.

3. **Георгий Леквеишвили**, родился в 1989 году в г.Кутаиси. Закончил физ-мат школу им. Размадзе в г.Кутаиси. Студент третьего курса ТбГУ им.И.Джавахишвили. Призер IOI. Неоднократный призер республиканских олимпиад по информатике среди школьников.

Третий призер Открытого чемпионата Украины 2006 года, Открытого чемпионата Москвы 2008 года, Олимпиады им. Лебедева и Глушкова 2007 года, Кубка Векуа 2007 и 2008, второй призер Олимпиады им. Лебедева и Глушкова 2008 года, Чемпион Грузии, Украины и Южного Кавказа 2008 года. Обладатель дипломов Открытого Кубка и Командного чемпионата РФ по программированию среди студентов. Обладатель желтого рейтинга сайта TopCoder. Хобби - спорт. Холост.

4. **Теодор Заркуа**, родился в 1951 году в г.Тбилиси. Закончил мех.-мат. Тбилисского ГУ. Служил офицером двухгодичником на Дальнем Востоке. Преподает в Вузах с 1975 года. Имеет опыт преподавания в школах. Тренирует команды программистов ТбГУ с 1999 года. В 1980-81 годах участвовал в работах по адаптации ОС ДИСПАК на МВК “Эльбрус” в НИИ ТМ и ВТ АН СССР. Несколько лет был директором департамента ИТ АО “Банк Грузии”. В данный момент директор международной Олимпиады по программированию на Кубок Векуа, Открытого чемпионата Южного Кавказа, а также полуфинала чемпионата Мира по программированию среди Вузов для Южно Кавказского подрегиона. Основатель и Президент Ассоциации Поддержки Программирования (Грузия). Имеет 17 научно-педагогических публикаций, среди них - один учебник по основам программирования. Имеет спортивные разряды по гандболу, футболу и шахматам. Женат, трое детей. Хобби - музыка.

Теоретический материал.

Некоторые способы обработки функциональных выражений

Как известно, для облегчения автоматизации обработки алгебраических выражений используются бесскобочные способы их записи, называемые польскими в честь их автора, польского математика Лукасевича. Различают два вида польских записей – постфиксный (когда знак операции записывается непосредственно после опреандов) и префиксный (когда знак

операции записывается непосредственно перед соответствующими операндами). Например, выражению *(a+b)*c-d/(e-f)* соответствует постфиксная запись *ab+c*def-/-* и префиксная запись *-*+abc/d-ef*.

В первой записи *ab+* сответствует *a+b*, далее рассматривая *ab+* как операнд, замечаем, что *ab+c** соответствует *(a+b)*c*. Далее *ef-* соответствует *e-f*, дальше *def-/* соответствует *d/(e-f)* и в итоге, рассматривая уже полученные *ab+c** и *def-/* как операнды, получаем, что *ab+c*def-/-* соответствует *(a+b)*c-d/(e-f)*. Аналогично можно убедиться в том, что приведенная префиксная запись соответствует исходному выражению.

Постфиксная форма записи существенно облегчает вычисление значения выражения, а префиксная форма также существенно облегчает автоматизацию функциональных преобразований выражения. И так как существуют задачи, в которых возникает необходимость и в вычислении выражения, и в осуществлении его функциональных преобразований, актуальна задача преобразования выражения из одной бесскобочной формы в другую.

Постфиксная форма записи выражения в удобном виде содержит всю информацию, необходимую для вычисления значения этого выражения. Многим, наверное, известен алгоритм вычисления выражения по его постфиксной форме. Вспомним его:

1. Взять пустой стек.
2. Взять очередной элемент выражения. Если это невозможно, т.е. выражение исчерпалось, перейти к пункту 5.
3. Если очередной элемент является операндом, то поместить его в стек и перейти к пункту 2.
4. Т.к. очередной элемент является обозначением операции, то считать из стека соответствующее количество записей (операндов), выполнить над ними эту операцию, учитывая, что операнды считывались в обратном порядке (скажем, при делении, сначала делитель, а потом делимое), поместить результат в стек и перейти к пункту 2.
5. Взять результат из стека, завершить выполнение операции.

Обозначим этот алгоритм символом A.

Определение 1. Сопряженным к A назовем алгоритм, который получается из A заменой подчеркнутого текста “в обратном порядке” на текст “в прямом порядке” и обозначим его A^*.

Отметим очевидную идентичность сложности алгоритмов A и A^*. Отметим, кроме того, свойство сопряженности, применительно к алгоритму A

совершенно очевидно вытекающее непосредственно из его определения: $(A^*)^*=A$.

Определение 2. Скажем, что постфиксное выражение w является сопряженным к постфиксному выражению v, если имеет место соотношение:

$$A^*(w) = A(v). \tag{1}$$

Если w является сопряженным к v, т.е. если имеет место (1), то запишем:

$$w = v^* \tag{2}$$

Совершенно очевидно, что если имеет место (1), то тогла имеет место также и $A(w)=A^*(v)$, а отсюда $w^*=v$.

Т.е. понятие сопряженности применительно к постфиксным записям оказывается взаимообратимым. Тогда для любого постфиксного выражения v справедливо равенство:

$$(v^*)^* = v \tag{3}$$

Обозначим символом C функцию преобразования постфиксного выражения к его сопряженному (Conjugate - сопряженный). Тогда, очевидно, для любого постфиксного выражения v по определению будем иметь: $C(v)=v^*$. Очевидно, что для любого постфиксного выражения существует сопряженное ему выражение, которое также является постфикстным. Очевидно также, что для любого постфиксного выражения v имеет место соотношение:

$$C(C(v)) = v. \tag{3*}$$

Сформулируем алгоритм преобразования постфиксного выражения к сопряженному:

1. Взять пустой стек.
2. Взять очередной элемент выражения. Если это невозможно, т.е. выражение исчерпалось, перейти к пункту 5.
3. Если очередной элемент является операндом, то поместить его в стек и перейти к пункту 2.
4. Т.к. очередной элемент является обозначением операции, то считать из стека соответствующее количество элементов (операндов), сформировать цепочку состоящюю из считанных элементов стека и приформированного в конце знака данной операции, учитывая, что элементы считывались в прямом порядке, поместить результат в стек и перейти к пункту 2.

5. Взять результат из стека, завершить выполнение операции.

Например, *C(ab-) = ba-*, *C(ab-c/) = cba-/*.

Если символом R (Reverse — перемена) обозначить преобразование реверсирования выражения, т.е. перестановки его элементов задом — наперед, а символом P обозначить функцию преобразования постфиксного выражения в префиксное, тогда будет верно следующее утверждение.

Теорема 1. Для любого постфиксного выражения v имеет место соотношение:

$$C(v) = R(P(v)). \tag{4}$$

Доказательство.Поведем доказательство по индукции относительно n — количества операций в выражении v. Когда количество $n=0$, справедливость (4) очевидна.

Допустим, что наше утверждение верно для всех n, у которых $n < k$, где $k \geq 1$. Покажем тогда, что оно обязательно будет верно и для таких v, у которых $n=k$. Действительно, пусть v - некоторое постфиксное выражение, для которого $n=k$. Обозначим символом @ операцию, расположенную в самом конце v (постфиксное выражение, содержащее хотя бы одну операцию обязательно заканчивается знаком операции). Если @ — одноместная операция, тогда $v=w@$, где w — постфиксное выражение, для которого $n < k$. Следовательно, для w по нашему допущению доказываемая теорема верна. Но тогда можем написать:

C(v)=C(w@)=C(w)@=R(P(w))@=R(@P(w))=R(P(w@))=R(P(v)).

Если же @ - многоместная операция, то тогда будем иметь: $v = w_1, \ldots w_m@$, где m > 1 и где каждое w_i представляет собой постфиксное выражение, удовлеворяющее по допущению доказываемой теореме. Поэтому можем написать:

C(v)=C(w_1,... w_m@)=C(w_m) ... $C(w_1)$@=R($P(w_m)$) ... $R(P(w_1))$@=R(@P(w_1) ... P(w_m))=R(P(v)).

Этим данная теорема полностью доказана.

Следствие 1. Для любого постфиксного выражения v имеет место соотношение:

$$P(v) = R(C(v)) \tag{5}$$

Действительно, (5) получим непосредственно применив к обеим частям соотношения (4) преобразование R и воспользовавшись очевидным соотношением $R(R(w))=w$, где w — любая последовательность.

А так как $C(v)=v^*$, то данная теорема подсказывает способ преобразования постфиксного выражения в префиксное.

А именно — для получения из постфиксного выражения соответствующего префиксного выражения, необходимо из исходного постфиксного выражения получить соответствующее сопряженное выражение, а затем отреверсировать его.

Преобразование, обратное преобразованию P обозначим P^{-1}. Таким образом, преобразование P^{-1} примененное к префиксному выражению, дает в результате соответствующее постфиксное выражение.

Следствие 2. Для любого префиксного выражения x имеет место соотношение:

$$P^{-1}(x) = C(R(x)). \tag{6}$$

Действительно, применив к обеим частям соотношения (4) преобразование C и воспользовавшись формулой (3*), получим:

$$v = C(R(P(v))). \tag{7}$$

Так как v — любое постфиксное выражение, можно обозначить $x=P(v)$, а тогда

$v = P^{-1}(x)$ и подставляя в (7) немедленно получаем (6).

Теперь из (6) делаем следуюший вывод. Для того, чтобы преобразовать префиксное выражение в постфиксное, необходимо взять сопряженное от его отреверсированного вида.

Следствие 3. Если v – постфиксное выражение, то:

$$R(v) = P(v^*). \tag{8}$$

Действительно,(8) немедленно получаем подставляя в (5) v^* на место v.

Следствие 4. Если x – префиксное выражение, то:

$$R(x) = (\mathrm{P} - 1(x))^*. \tag{9}$$

Доказательство. (9) получается применением преобразования C к обеим частям (6).

Внимательно присмотримся к последним двум результатам. Оказывется обработка постфиксного выражения справа налево (т.е. фактически без всякого предварительного преобразования !) равносильна работе с префиксным выражением, которое получается из сопряженного ему выражения.

Совершенно аналогично, обработка префиксного выражения справа налево равносильна работе с выражением, сопряженным соответствующему постфиксному.

И все это наталкивает на следующюю схему обработки выражения в тех случаях, когда имеется необходимость иметь в активном состоянии соответствующие друг другу постфиксную и префиксную записи.

Пусть нам необходимо время от времени осуществлять некоторые функциональные преобразования исходного выражения, после которого оно попадает в блок, который использует значения этого выражения для определенных значений аргументов, затем выражение вновь подвергается функциональным преобразованиям, затем снова используется (уже измененный вид) и т.д. несколько раз. На первый взгляд кажется, что необходимо максимально оптимизировать фрагмент, осуществляющий преобразования выражения из постфиксной формы в префиксную и наоборот.

Однако, вышесказанное наводит на мысль вообще обойтись без преобразований. Действительно, имея префиксное выражение, мы, де факто, одновременно имеем выражение, сопряженное соответствующему потсфиксному. Для вышеобозначенных целей вместо постфиксного выражения мы всегда можем использовать соответствующее сопряженное. Для этого достаточно, скажем, вместо алгоритма A применить алгоритм A^* совершенно идентичный по сложности с алгоритмом A. Более того, так как A^* предусматривает получение операндов в естественной последовательности, то в некоторых случаях это позволит использовать сокращенную схему вычисления выражений. И так как функциональные преобразования удобнее осуществлять в терминах префиксного выражения, нам остается сконцентрировать наши усилия на реализации этих функциональных преобразований только над наиболее удобным для этого — префиксным выражением. Имея префиксное выражение, при необходимости, мы всегда сможем обработать его справа налево алгоритмом A^* и получить тот же эффект, кторый дало бы нам соответствующее постфиксное выражение, без необходимости выполнить преобразование для получения этого самого постфиксного выражения!

В заключение, дабы подкрепить вышесказанное приведем несколько примеров, подтверждающих утверждение об удобстве префиксной формы для реализации функциональных преобразований.

Посмотрим, как выглядят некоторые общеизвестные формулы дифференцирования в обычных терминах и терминах префиксной записи (перегружаемой операцией взятия производной будем считать операцию !):

$(const)' = 0$	$!\ const = 0$
$(x)' = 1$	$!\ x = 1$
$(x_n)' = n * x^{n-1}$	$!\ \hat{}x\ n = *\ n\hat{}x - n1$
$(a^x)' = a^x * ln(a)$	$!\ \hat{}a\ x = *\hat{}a\ x\ ln\ a$
$(u + v)' = u' + v'$	$!+ u\ v = +\ !\ u\ !\ v$
$(u * v)' = u' * v + u * v'$	$!\ *u\ v = +\ *\ !\ u\ v\ *u\ !\ v$
$(sin(x))' = cos(x)$	$!\ sin\ x = cos\ x$
$(u/v)' = (u' * v - u * v')/v^2$	$!\ /\ u\ v = /-\ *\ !\ u\ v\ *\ u\ !\ v\hat{}v\ 2$

Здесь для обозначения операции возведения в степень был использован знак ^.

Задачи и разборы

Задача A. Максимальное число

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Дана таблица цифр размером NxN. Пометим в ней N элементов таким образом, чтобы из каждой строки и каждого столбца был помечен ровно один элемент.

Далее составим из всех помеченных цифр число так, что его первой цифрой взята цифра, помеченная в первой строке, второй – во второй, и так далее. Например, из помеченных в следующей таблице цифр мы таким образом получим число 4307:

2	<u>4</u>	2	1
<u>3</u>	4	2	0
1	3	<u>0</u>	0
8	1	8	<u>7</u>

Определите, какое максимальное число можно получить вышеописанным способом из данной таблицы.

Ограничения

$2 \le N \le 100$

Цифры в таблице от 0 до 9 включительно.

Формат входного файла

Первая строка содержит целое число N. Далее следует сама таблица - каждая из следующих строк содержит N цифр без пробелов или каких-либо других разделителей.

Формат выходного файла

Вынесите максимальное число, которое можно получить из данной таблицы. Не пропускайте ведущих нулей (см. пример).

Пример

a.in	a.out
4 2421 3420 1300 8187	4308
3 000 222 987	029

Разбор задачи A. Максимальное число

Если бы в задаче гарантировалась единственность каждой цифры в каждой из строк (хотя ограничение на N в таком случае было бы максимум 10), решением задачи было бы пометить клетку с наибольшей цифрой в первой строке, а в каждой следующей строке метить клетку с наибольшей цифрой из всех, находящихся в непомеченных столбцах. Так как в этой задаче цифры в строках могут повторяться, при рассмотрении строки невозможно однозначно решить, какую из клеток с наибольшей цифрой пометить. Поэтому мы будем действовать так: в первой строке пометим все клетки, содержащие наибольшую цифру в этой строке. В каждой следующей строке попытаемся пометить клетки с наибольшей цифрой соответствующей строки. Скажем, мы рассматриваем строку с номером i. Нам нужно проверить, можно ли выбрать из первых i строк ровно i помеченных клеток из различных столбцов. Если это невозможно, значит выбрать в строке i наибольшую цифру так, чтобы при этом предыдущие $i-1$ были наибольшими возможными, нельзя. Тогда нужно проверить следующую после наибольшей цифру этой строки, и так далее. Для одной из цифр выбрать i клеток обязательно получится. Таким образом, после того как

будет выбрана цифра в последней строке, мы будем иметь наибольшее возможное число из этой таблицы.

Теперь расскажем о том, как проверить возможность выбора i помеченных клеток из различных столбцов. Построим двудольный граф с i вершинами в левой доле и N вершинами во второй. Вершины левой доли будут соответствовать строкам, а правой - столбцам. Ребро между вершинами A левой доли и B правой доли проведём в том случае, если помечена клетка на пересечении строки A и столбца B. Выбрать ровно i помеченных клеток из различных столбцов можно тогда и только тогда, когда существует паросочетание в построенном графе с i парами.

Задача B. Двоичные строки

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Двоичной (бинарной) называют строку, которая состоит только из символов “0” и “1”. Суффиксом строки называют строку, которая получается, если стереть из оригинала первые $K \geq 0$ символов. Например, суффиксами строки “1001” являются строки “1001”, “001”, “01”, “1” и пустая строка.

Даны N двоичных строк. Найдите, сколько среди них пар таких, что одна из строк в паре является суффиксом второй.

Ограничения

$2 \leq N \leq 100000$
Суммарная длина всех строк не превосходит 200,000.

Формат входного файла

Первая строка содержит целое число N. Каждая из следующих N строк содержит по одной непустой двоичной строке.

Формат выходного файла

Единственное целое число – количество искомых пар.

Пример

b.in	b.out
5 0 1 10 0 001	4

Разбор задачи B. Максимальное число

Очевидно, что если перевернуть входные строки (т.е. прочитать их задом наперёд), задачу можно переформулировать так: найти количество пар строк, в которых одна из строк является префиксом второй. Для её решения можно воспользоваться структурой под названием trie-дерево. В данном случае, дерево будет бинарное, так как алфавит состоит только из двух символов, 0 и 1. В каждой вершине дерева нужно запоминать, сколько строк прошли по этой вершине *(X)*, и сколько строк в этой вершине закончились *(Y)*. С их помощью можно без сортировки строк вычислить ответ задачи. Число X вершины, в которой некоторая строка закончилась, есть количество уже обработанных строк, префиксом которых она является. Число же Y в каждой вершине пути для некоторой строки есть количество обработанных строк, которые являются префиксами данной и закончились в этой вершине. Построение trie-дерева и одновременное вычисление ответа можно выполнить в $O(L)$, где L – суммарная длина строк.

Задача C. Стрекоза и муравей

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Сложные и многогранные взаимоотношения между стрекозой и муравьем отражены отнюдь не только в известной басне Крылова. В частности, согласно одноименной грузинской народной сказке, стрекоза выручает муравья, попавшего в реку до того, как тот успел научиться плавать. Кстати, о том как обстоят дела у муравья в этом смысле в данный момент, народный эпос пока умалчивает. Рассказ о том, как все же стрекоза сумела выручить муравья из беды, заслуживает отдельной задачи. Мы же проследим за событиями с того момента, когда спасенный муравей решил

пригласить своего спасителя в лучшую хинкальную леса, в котором они обитали. В момент, когда муравью пришла в голову такая мысль, друзья находились на краю опушки, которую можно представить как прямоугольный участок размером *M*x*N*. Каждая клетка этого участка характеризуется своей высотой от уровня моря. При этом, друзья находятся в клетке (1,1), а вожделенная хинкальная расположена на другом конце опушки в клетке *(M,N)*. Каждым очередным шагом друзья могут попасть в любую из соседних клеток, расположенных внутри опушки. Две квадратные клетки считаются соседними, если у них есть общая сторона.

Перед друзьями стала задача подобрать такой маршрут, чтобы возможно меньше устать. Но как им быть, если для стрекозы предпочтителен маршрут, проходящий по меньшему количеству клеток, а для муравья – маршрут, в котором минимальна максимальная высота, которую ему придется взять. Начальная и конечная клетки входят в маршрут. Так как по версии грузинской народной сказки и стрекоза, и муравей хорошо воспитаны, то каждый из них настаивает на варианте, объективно предпочтительном для друга. И постольку поскольку мы пока еще не знаем, кто оказался более настойчивым, нам необходимо быть готовыми помочь друзьям при подборе маршрута в обоих возможных вариантах. Таким образом, нам необходимо определить длину пути, т.е. количество пройденных клеток, с учетом начальной и конечной, а также максимальную высоту, взятую по пути друзьями при условии, что:

- был подобран маршрут, наиболее удобный для стрекозы, а в случае, если таких оказалось несколько, среди них был выбран маршрут, наилучшим образом отвечающий интересам муравья;
- был подобран маршрут, наиболее удобный для муравъя, а в случае, если таких оказалось несколько, среди них был выбран маршрут, наилучшим образом отвечающий интересам стрекозы.

Ограничения

M и *N* – целые числа, причем $2 \leq M, N \leq 500$. Высота каждой клетки – это целое число, принадлежащее диапазону 1. . . 1,000,000,000.

Формат входного файла

Первая строка содержит значения чисел *M* и *N*. Каждая из следующих *M* строк содержит по *N* чисел. *j*-ое число *i+1*-ой строки соответствует высоте клетки *(i, j)*.

Формат выходного файла

Две строки, каждая из которых содержит по два целых числа. В первой строке выводятся длина и максимальная высота маршрута для первого случая, а во втором соответствующие данные для второго случая.

Пример

c.in	c.out
4 5 2 3 2 1 2 1 4 1 4 2 2 1 2 3 2 1 2 1 4 1	8 3 12 2

Пояснение

Стрекозу устраивает, например, маршрут (1,1)–(1,2)–(1,3)–(1,4)–(1,5)–(2,5)–(3,5)–(4,5), в котором учтены и интересы муравья. Наилучшим маршрутом для муравья же был бы (1,1)–(2,1)–(3,1)–(3,2)–(3,3)–(2,3)–(1,3)–(1,4)–(1,5)–(2,5)–(3,5)–(4,5).

Разбор задачи C. Стрекоза и муравей

Рассмотрим сначала, как найти наилучший путь для стрекозы. Очевидно, что на поле размером *M*x*N*, где можно пройти по каждой клетке, кратчайшим путём из *(1,1)* в *(M,N)* является любой путь длины *M+N-1* – пройти *M-1* клеток в направлении увеличения первой координаты и *N-1* клеток в направлении увеличения второй. Нас интересует такой кратчайший путь, на котором высота наибольшей пройденной клетки будет как можно ниже. Заметим, что в любую клетку *(A, B)* можно попасть только из клеток *(A-1, B)* или *(A, B-1)*. Поэтому можно написать простое рекуррентное соотношение:

$$ANS\left[1,1\right] = H\left[1,1\right]$$

$$ANS\left[A,B\right] = MAX(H\left[A,B\right], MIN(ANS\left[A-1,B\right], ANS\left[A,B-1\right])$$

Здесь $H\left[i,j\right]$ – высота клетки *(i, j)*, а $ANS\left[i,j\right]$ – минимальная возможная высота пройденных на кратчайшем пути к клетке *(i, j)* клеток. Подразумевается, что $ANS\left[0,i\right]$ и $ANS\left[i,0\right]$ равны бесконечности для всех i. С помощью данной рекуррентной формулы можно найти значение $ANS\left[M,N\right]$, которое и является ответом на первую часть задачи. Вычисление значений $ANS\left[\right]$ нужно производить только один раз, а учитывая, что для любой клетки нам требуются только значения клеток с меньшими координатами,

это можно сделать просто в двойном цикле. Сложность *O(NM)*.

Теперь рассмотрим нахождения наилучшего пути для муравья. Скажем, что минимальная возможная высота, которую ему придётся покорить, равна *X*. Тогда очевидно, что не существует пути из *(1,1)* в *(M,N)*, который проходит только по клеткам с высотой меньше *X*. Следовательно, можно предложить следующий алгоритм для нахождения *X*: перебрать *X* двоичным поиском в пределах от минимального до максимального возможного его значения (при данных в задаче ограничениях это 1 и 1,000,000,000, хотя для каждого теста поисковый интервал можно уменьшить в зависимости от значений высот клеток) и найти первое такое значение *X*, для которого существует какой-либо путь из *(1,1)* в *(M,N)*. Проверку можно осуществить с помощью поиска в ширину. Также после определения *X* с помощью поиска в ширину можно найти и наикратчайший из тех путей, которые используют клетки с высотой не более *X*. Итоговая сложность алгоритма - $O(lgMAX * NM)$, где *MAX* - максимальная разница между высотой клеток.

Задача D. Кладоискатели

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

В результате длительных поисков кладоискатели обнаружили огромное поле, на котором есть *N* точек с золотыми кладами. У предводителя есть карта, на которой помечены координаты и количество золота в каждом из кладов. Поле представляет собой множество тех точек *(x, y)* прямоугольной координатной плоскости, ординаты которых положительны. Лагерь кладоискателей расположен по всей оси *X*. Координаты точек с кладами - целые числа.

У предводителя есть такой секретный план: он подойдёт к какой-либо точке поля с целыми координатами и начнёт двигаться к лагерю. Чтобы его движение не показалось подозрительным другим кладоискателям, из каждой точки (x,y) он будет передвигаться только в точку *(x-1, y-1)*, *(x, y-1)* или *(x+1, y-1)*. Проходя точку с кладом, он будет незаметно забирать себе из него всё золото. Когда предводитель достигнет оси *X*, он остановится. Найдите, какое максимальное количество золота, которое сможет присвоить предводитель кладоискателей, действуя согласно своему плану.

Ограничения

$3 \leq N \leq 50000$

Координата *x* каждого из кладов удовлетворяет условию $-1,000,000,000 \leq x \leq 1,000,000,000$.

Координата y каждого из кладов удовлетворяет условию $1 \leq y \leq 1,000,000,000$.

Количество золота в каждом из кладов - целое число в диапазоне $[1, 10^9]$.

Формат входного файла

Первая строка содержит целое число *N*. Каждая из следующих *N* строк содержит числа x_i, y_i, c_i - координаты *i*-ого клада и количество золота в нём, соответственно.

Формат выходного файла

Единственное целое число - максимальное возможное количество золота, собранного предводителем.

Пример

d.in	d.out
5 0 5 2 2 4 3 -1 3 2 0 2 1 1 1 3	8

Разбор задачи D. Кладоискатели

Для начала проведём некоторые преобразования. А именно, заменим каждую точку *(x, y)* на тот сегмент на оси *X*, в который можно попасть, двигаясь по правилам задачи из точки *(x, y)*. Тогда точке *(x, y)* будет соответствовать сегмент $[x - y, x + y]$. Суммой сегмента назовём количество золота в соответствующей ему точке.

Если точка *(x2, y2)* достижима из точки *(x1,y1)*, тогда сегмент $[x1 - y1, x1 + y1]$ будет содержать сегмент $[x2 - y2, x2 + y2]$. То есть задача свелась к довольно известной задаче – найти последовательность вставленных сегментов с максимальной общей суммой.

Отсортируем полученные сегменты по убыванию координаты правого конца (в случае равенства, по возрастанию координаты левого). Теперь можно с помощью динамического програмирования посчитать ответ задачи:

$$\mathbf{ANS}\,[i] = MAX(\mathbf{ANS}\,[j] \mid j < i \ AND \ \mathbf{L}\,[j] < \mathbf{L}\,[i]) + \mathbf{S}\,[i]$$

Здесь $ANS\,[i]$ – максимальная из возможных общих сумм последовательностей вставленных сегментов, заканчивающихся на i-ом сегменте, $L[i]$ – координата левого конца i-ого сегмента, а $S[i]$ – сумма i-ого сегмента. Правда, при ограничении, данном в задаче, подсчитывать значения *ANS* линейно не получится, поэтому нужно “сжать” координаты левых концов до чисел от 1 до N и затем применить подходящую структуру данных, например Binary Indexed Tree или деревья интервалов.

Задача E. Однажды в Китае

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Скоро увидит свет новая компьютерная игра “Однажды в Китае”. Игра заключается примерно в следующем: один джигит овладевает исскуством кунг-фу и начинает обходить разные бандитские логова, где он избивает плохишей и попутно забирает у них награбленные деньги. Уровень владения кунг-фу выражается неким неотрицательным рациональным числом.

Всего в игре N бандитских логовов. i-тое логово имеет три показателя: Q_i, S_i и M_i. Число Q_i указывает минимальное владение кунг-фу, которое должно быть у джигита, чтобы он смог ограбить бандитов в логове i. Если его владение кунг-фу менее Q_i, ему туда лучше не соваться. S_i - это сумма (в юанях), которую сможет забрать джигит из i-ого логова, если его владение кунг-фу Q_i. Если же его кунг-фу лучше, то он может выбить у бандитов и побольше деньжат. А конкретно, если владение кунг-фу (обозначим его K) в диапазоне от Q_i до $Q_i * M_i$, то джигит заберёт у бандитов сумму $S_i * \frac{K}{Q_i}$. Если владение кунг-фу превосходит $Q_i * M_i$, джигит заберёт ровно $S_i * M_i$, потому что больше оттуда забирать нечего.

Звучит всё просто, но есть один подвох. Кунг-фу джигит должен обучаться у мастера Цзен, который за каждый час тренировки берёт 1 юань. А как известно, чем выше уровень ученика, тем дольше ему надо заниматься для повышения квалификации. Чтобы повысить уровень от X_1 до X_2, джигиту нужно $A * (X_2^2 - X_1^2)$ часов. Мастер Цзен может проводить тренировки

любой длительности и хочет, чтобы ему платили точно, но он может обучать в кредит, чтобы джигит вернул ему деньги после применения своих навыков.

Учитывая, что в начале игры джигит не владеет кунг-фу (т.е. его уровень 0) и его баланс на нуле, найдите максимально возможную прибыль (т.е. разницу между отбитыми у бандитов средствами и потраченной на тренировки суммой), которую он может получить.

Ограничения

$1 \leq N \leq 10000$

$0 \leq A \leq 10$, число дано не более чем с 3 знаками после запятой.

$1 \leq Q_i,\ S_i \leq 1000$

$1 \leq M_i \leq 10$

Числа M_i, S_i и Q_i - целые.

Формат входного файла

Первая строка содержит числа N и A.

Каждая из следующих N строк содержит числа Q_i, S_i и M_i.

Формат выходного файла

Одно число - максимальная прибыль джигита. Ваш ответ должен отличаться от правильного не более, чем на 0.000001 (1e-6).

Пример

e.in	e.out
1 0.15 5 10 3	6.6666666667

Разбор задачи E. Однажды в Китае

Первым делом, заметим, что так как можно обучаться в кредит, герой этой игры может сначала доучиться до некоторого уровня, а уже затем пойти добывать деньги, больше не проводя тренировок. Также заметим, что джигиту всегда выгодно ограбить все логова, которые он сможет при своём уровне владения кунг-фу. Рассмотрим функцию прибыли джигита в зависимости от его уровня владения кунг-фу $W(K) : R+ \rightarrow R$. В каждой из точек Q_i и $Q_i * M_i$ эта функция имеет разрыв, а на интервалах между соседними точками такого вида функция имеет вид $W(K) = -A * K^2 + B * K + C$, где A – параметр, данный в условии задачи, а коефициенты B и C вычисляются следующим образом:

- Для всех логовов i таких, что $Q_i \leq K < Q_i * M_i$, $B(K) = B(K) + \frac{S_i}{Q_i}$.
- Для всех логовов i таких, что $Q_i * M_i \leq K$, $C(K) = C(K) + S_i * M_i$.

Очевидно, что на каждом из упомянутых интервалов график функции представляет собой фрагмент параболы с ветвями, направленными вниз. Максимум функции на каждом из этих интервалов находится или в вершине параболы, если она попадает в этот интервал, или в одном из концов интервала, а глобальный максимум функции – это наибольший из этих максимумов.

Чтобы эффективно проверить значения функции *W(K)* в каждой из вышеупомянутых точек, следует рассмотреть все точки разрыва в порядке возрастания координаты и изменять коефициенты функции соответственно. Тогда итоговая оценка сложности алгоритма будет *O(NlgN)*.

Задача F. Возвращение Супермена

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Все знают, кто такой Супермен? На случай, если нет — это такой мужик из другой галактики, которого кулаком не ударишь, да и пули не берут. Короче неуязвимый он. Ладно, почти неуязвимый. Есть вещество, которое оказывает на него губительное воздействие — куски зелёного метеорита с его родной планеты. Когда они находятся поблизости, Супермен теряет свои поразительные способности. Коварные враги Супермена ранили его пулей из зеленого метеорита. Теперь он медленно умирает. Единственное спасение для него — добраться в Крепость, его единственную связь с родной планетой. Там из него извлекут пулю и реабилитируют.

Рассмотрим нашу задачу на плоскости. Супермен был ранен в точке *S(x1,y1)*. Крепость находится в точке *F(x2,y2)*. Начальная скорость Супермена *V* м/с. Каждую секунду вследствие ранения она уменьшается на *A* единиц. Учитывая, что любое передвижение ещё более расходует силы Супермена, каждый пройденный им метр уменьшает скорость ещё на *B*

единиц. Причём изменение скорости вследствие обоих факторов происходит непрерывно. Если скорость достигнет нулевой отметки, значит Супермен больше не может двигаться.

Кроме всех этих бед, где-то вокруг есть залежь того же зелёного метеорита (будем рассматривать её как точку). В зависимости от количества вещества, у неё есть радиус действия. Если раненый Супермен окажется внутри этого радиуса, он погибнет.

Нам даны *x1*, *y1*, *x2*, *y2* — координаты Супермена в начальный момент и координаты Крепости. Далее, даны числа *X*, *Y*, *R* — координаты залежи метеоритов и радиус её действия. В конце даны числа *V*, *A* и *B*.

Если Супермен не сможет добраться до Крепости, вынести “-1” (quotes for clarity). В противном случае, вынести минимальное время, за которое он это сделает.

Гарантируется, что если Супермен не может добраться до крепости, то он остановится на расстоянии не менее 0.000001 (1e-6) от неё.

Ограничения

Числа *x1*, *y1*, *x2*, *y2*, *X*, и *Y* — целые и по абсолютному значению не превосходят 50. Число *R* находится в диапазоне от 1 до 50, включительно. *V*, *A* и *B* даны с точностью до двух знаков и удовлетворяют $0 < V, A \leq 50$; $0 \leq B \leq 50$.

Точки *S* и *F* различны.

Формат входного файла

Первая строка содержит числа *x1*, *y1*, *x2*, *y2*.

Вторая строка содержит числа *X*, *Y*, *R*.

Третья строка содержит числа *V*, *A*, *B*.

Формат выходного файла

Если Крепость достижима, вынести минимальное время, за которое Супермен доберётся до неё. Ваш ответ не должен отличаться от правильного более, чем на 0.001. Если Крепость недостижима, вынести -1.

Пример

f.in	f.out
10 1 2 2 3 4 1 5 0.1 0.4	2.745791

Разбор задачи F. Возвращение Супермена

Очевидно, что время, за которое можно добраться из точки *S* в точку *F*, зависит только от длины пути и на него не влияет выбранная траектория движения. Поэтому задачу можно поделить на две части: нахождение наикратчайшего пути, не пересекающего окружность, а затем нахождение времени, за которое такой путь можно пройти.

Рассмотрим первую часть задачи. Для начала заметим, что если хотя бы одна из точек *S* и *F* находится внутри окружности, никуда Супермен добраться уже не сможет и ответ задачи -1.

Если отрезок *SF* не пересекает окружность, его длина и есть кратчайший путь. В противном случае поступим следующим образом: проведём касательные из *S* и *F* к окружности. Кратчайшим путём будет пройти по касательной из *S*, потом по дуге окружности и затем по касательной из *F*.

Таким образом, мы имеем кратчайший путь. Скажем, его длина *D*. Теперь перейдём ко второй части задачи. Наша скорость описывается уравнением:

$$V(t) = V0 - at - bS(t)$$

Отдельно рассмотрим вид уравнений для скорости и пути, если *b*=0:

$$V(t) = V0 - at$$

$$S(t) = \int V(t)dt = V_0 t - \frac{at^2}{2}$$

Максимальное расстояние, которое пройдет Супермен, равняется $S(\frac{V_0}{a})$. Сравнивая это расстояние с D, можно узнать, доберётся ли герой до Крепости, и если да, то найти соответствущее время из уравнения $D = V_0 t - \frac{at^2}{2}$.

Если же b>0, мы имеем $S'(t) = V0 - at - bS(t)$, что является дифференциальным уравнением.

Умножим обе части уравнения на величину e^{bt}:

$$S'(t)e^{bt} + bS(t)e^{bt} = V_0 e^{bt} - ae^{bt}t$$

$$\left[S(t)e^{bt}\right]' = V_0 e^{bt} - ae^{bt}t$$

$$S(t) = e^{bt}(C + \int V_0 e^{bt}dt - a\int te^{bt}dt$$

$$S(t) = e^{-bt}(C + \frac{V_0}{b}e^{bt} - \frac{a}{b}e^{bt}(t - \frac{1}{b}))$$

Чтобы вычислить константу *C*, используем начальное условие: в нулевой момент времени пройденное растояние равняется 0.

$$0 = C + \frac{V_0}{b} + \frac{a}{b^2}$$

В итоге, получаем следующий вид для *S(t)*:

$$S(t) = -e^{-bt}(\frac{V_0}{b} + \frac{a}{b^2}) + \frac{V_0}{b} - \frac{a}{b}t + \frac{a}{b^2}$$

Подставляя S(t) в выражение для V(t) или попросту дифференцируя его, получаем:

$$V(t) = e^{-bt}(V_0 + \frac{a}{b}) - \frac{a}{b}$$

Теперь сначала определим, пройдет ли Супермен вообще путь длины *D*. Для этого найдем, какой максимальный путь он может пройти. Так как скорость уменьшается, точка, в которой она станет равна нулю, является точкой максимума пройденного пути.

$$0 = e^{bt^*}(V_0 + \frac{a}{b}) - \frac{a}{b}$$

$$t^* = \frac{\ln(\frac{V_0 b}{a} + 1)}{b}$$

Если *S(t*)<D*, он просто не дотянет до точки *F*, соответственно ответ задачи -1. В противном случае, переберем двоичным поиском t, чтобы найти такое $t\hat{}$, что $S(t\hat{}) = D$ с нужной точностью.

Задача G. Нелюбимые цифры

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Новый руководитель организации обнаружил, что его предшественник, по одному ему известным причинам, для нумерации официальных документов принципиально не использовал числа, десятиричные записи которых содержали некоторые цифры. Причем в разные годы обструкции подвергались разные комплекты цифр.

В качестве начального номера в каждом году, предыдущий руководитель брал минимальное неотрицательное число, не содержащее отвергнутых им в данном году цифр.

При нумерации каждого следующего документа, в тех случаях, когда следующий номер содержал отвергнутую цифру, он просто пропускал это число. И так до тех пор, пока очередное число не оказывалось свободным от нежелательных ему цифр. Например, если отвергались 8, 7, 9, 5, 1, то первые несколько документов этого года имели следующие номера: 0, 2, 3, 4, 6, 20, 22, 23, 24, 26, 30, 32, 33, ...

И так как предшественник руководил организацией довольно долго и накопилось изрядное количество перенумерованных им документов, у нового руководителя возникла потребность в программе, которая для заданного комплекта отвергнутых цифр по порядковому номеру документа, отсчитанному с нуля, быстро определит номер, который был официально ему присвоен.

Ограничения

Общее количество отвергнутых цифр не менее 1 и не более, чем 8. Порядковый номер искомого документа не меньше нуля и не превосходит 1,000,000,000.

Формат входного файла

Входной файл содержит две непустые строки. В первой строке через пробел перечислены нелюбимые цифры (их общее количество от одной до восьми включительно). Во второй строке дан порядковый номер искомого документа.

Формат выходного файла

Выходной файл содержит единственное число - номер запрошенного документа, т.е. официальный номер документа, порядковый номер которого указан во входном файле.

Пример

g.in	g.out
8 7 9 5 1 8	24

Разбор задачи G. Нелюбимые цифры

Вначале замечаем, что нас интересуют цифры, которые разрешено использовать (назовем их допускаемыми).

Если среди этих цифр есть 0, то мы имеем дело с позиционной системой счисления, основание которой равно количеству допускаемых цифр. В этой системе в качестве цифр используются символы, не обязательно совпадающие по написанию с символами, используемыми в качестве цифр в стандартной позиционной системе счисления с таким же основанием. Однако имеется возможность установить взаимно од-нозначное соответствие с числами такой системы. Например, если допускаются 3, 7, 6 и 0, то каждому номеру, проставленному предыдущим начальником, однозначно соответствует число в 4-ичной системе, которое получается из исходного заменой 3 на 1, 6 на 2 и 7 на 3. А так как числа стандартной 4-ричной системы составляют последова-тельность, заполняющую множество натуральных чисел полностью (т.е. без “дырок”), то остается лишь перевести заданный порядковый номер в стандартную позиционную систему счисления с основанием, равным количеству допускаемых цифр (промежуточную систему), а затем из полученной записи путем замены цифр получить искомый номер.

Если же среди допускаемых цифр нет цифры 0, то перед нами система без нулей. В качестве промежуточной системы счисления будем использовать нестандартную (без нуля) систему счисления с основанием, равным количеству допускаемых цифр. Эта система использует цифры, начиная с 1 до цифры, равной основанию системы и покрывает все целые числа, начиная с 1. Например, если допускаемые цифры 3, 8 и 4, то соответствующей промежуточной системой будет 3-ичная система с цифрами 1,2,3, в которой 1 соответствует 3, 2 соответствует 4, 3 соответствует 8.

При переводе в такую систему известным методом деленияавзятия остатка, следует учесть, что остаток, равный 0, заменяется на 3 и одновременно результат деления уменьшается на 1. А так как нас интересует номер, отсчитанный от 0, то перед началом процесса перевода следует исходное число увеличить на 1. Например, рассмотрим процесс получения 36-ого члена последовательности чисел с допускаемыми цифрами 3,4,8: делим 37 на 3, остаток 1, результат 12. Дальше делим 12 на 3. Так как в остатке 0, то заменяем его на 3, одновременно результат деления (4) заменяем на 3. Аналогично, снова получаем в “остатке” 3, а в качестве результата деления 0 (ведущий нуль). На этом процесс перевода в промежуточную систему завершается получением записи “331”, откуда заменой цифр получается ответ “883”.

Задача H. Шоколад

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Анна и Бека делят прямоугольную шоколадку размером MxN. Делают дети это так: они по очереди отламывают от одного из концов шоколадки кусок квадратной формы со стороной, равной меньшей из сторон шоколадки. Если в какой-либо момент остаётся шоколадка квадратной формы, ребёнок, который должен отламывать следующим, забирает всю оставшуюся часть и процесс разделения заканчивается. Анна начинает первой.

Рассмотрим пример. Скажем, сначала у детей была шоколадка размером 6x10. Первой отламывает Анна и забирает кусок размером 6x6. Бека отламывает от оставшейся шоколадки размером 6x4 кусок размером 4x4. Анна отламывает от оставшейся 2x4 кусок размером 2x2, и оставшийся кусок 2x2 Бека забирает целиком. В итоге, Анне достались куски с общей площадью 40, а Беке – с общей площадью 20.

Дано, шоколад какой площади достался в результате такого разделения Анне и какой – Беке.

Найдите начальные размеры шоколадки M и N, для которых доставшиеся детям площади были бы равны данным.

В случае нескольких решений, вынесите то, в котором M наименьшее. Если и таких несколько, вынесите то, в котором N наименьшее. Если таких M и N не существует, вынесите два числа -1.

Ограничения

Общие площади кусков шоколада, доставшиеся Анне и Беке в отдельности, не меньше 1 и не превосходят 1,000,000,000.

Формат входного файла

Первая строка содержит пару целых чисел. Первое из них – площадь шоколада, доставшегося Анне, а второе – площадь шоколада, доставшегося Беке.

Формат выходного файла

Если задача имеет решение, выводится пара чисел M и N, разделённых пробелом. В противном случае, выводятся два числа -1.

Пример

h.in	h.out
40 20	6 10
2 3	-1 -1

Разбор задачи H. Шоколад

Естественно, что начальная площадь шоколада равна сумме площадей, доставшихся детям, т.е. при данных ограничениях она не превосходит S=2,000,000,000. Меньшая из начальных сторон шоколада не может превосходить *sqrt(S)*, соответственно, если найти способ быстро определить площади, которые достанутся детям при данных начальных сторонах шоколада, задачу можно решить перебором меньшей начальной стороны и проверкой.

Если внимательно посмотреть на процесс, можно увидеть, что он аналогичен алгоритму Евклида для нахождения наибольшего общего делителя двух чисел. Оценка сложности алгоритма Евклида *O(logA*logB)*, поэтому итоговая оценка сложности алгоритма данной задачи *O(sqrt(A+B)*logA*logB)*.

Задача I. Похищение невесты

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Говорят, что высоко в горах (но, конечно, не в нашем районе) всё ещё сохранился красивый древний обычай — похищать невесту. Некий джигит как раз собрался это сделать. В распоряжении джигита есть прекрасный конь, его верный соратник в делах сердечных.

Для того, чтобы достойно совершить похищение, джигиту придётся хорошенько тренироваться в условиях, приближенных к реальным.

Для тренировок у джигитов есть специальное место. На нём выделены N пунктов, пронумерованные от 1 до N, и M дорожек. Согласно древнему обычаю, каждая из дорожек имеет единственное направление, в котором должен двигаться наездник. Двигаться по соответствующей дорожке в противоположном направлении опасно для жизни (все остальные джигиты воспринимают это как кровную обиду — со всеми вытекающими отсюда

последствиями). Также каждая из дорожек имеет сложность, выраженную натуральным числом.

Возможно существование нескольких дорожек между парой пунктов, или существование круговой дорожки, т.е. дорожки, для который начальный и конечный пункты совпадают.

Маршрутом назовём такую непустую последовательность дорожек, в которой каждая следующая дорожка (кроме первой) начинается в том пункте, где закончилась предыдующая. Полезным назовем маршрут, в котором сложность каждой дорожки (кроме начальной) строго выше предыдущей. Джигит должен избрать маршрут, который начинается и заканчивается в пункте 1. Естественно, в интересах дела, джигит должен пользоваться только полезными маршрутами.

Ваша задача — найти, сколько различных полезных маршрутов имеется в распоряжении джигита. Так как таких маршрутов может быть очень много, вынесите остаток при делении их количества на 1,000,000,000.

Ограничения

$1 \leq N \leq 50000$

$1 \leq M \leq 100000$

Сложность каждого маршрута — целое число от 1 до 30000.

Формат входного файла

Первая строка содержит числа N и M. Каждая из следующих M строк содержит целые числа A_i, B_i, C_i — начальный и конечный пункты i-ой дорожки и её сложность, соответственно.

Формат выходного файла

Единственное целое число — остаток от деления количества полезных маршрутов на 1,000,000,000.

Пример

i.in	i.out
4 6 2 1 4 1 4 5 4 1 6 2 3 2 3 2 3 1 2 1	5
2 4 1 2 1 1 2 1 1 2 1 2 1 2	3

Разбор задачи I. Похищение невесты

Учитывая, что пункты и дорожки образуют граф, мы будем называть пункты вершинами, дорожки рёбрами, а сложность дорожек весом ребра.

Для начала допустим, что веса ребёр — различные числа. Отсортируем все рёбра в порядке возрастания веса и представим, что вначале у нас имеется граф без ребер и мы будем их постепенно добавлять. Обозначим через $WAYS[i]$ имеющееся на данный момент количество полезных маршрутов, начинающихся в вершине 1 и заканчивающихся в вершине i. В пустом графе существует единственный маршрут, следующий из вершины 1 в вершину 1 (не содержащий ни одного ребра). Других маршрутов вначале нет.

Начнём добавление ребёр. При добавлении ребра, следующего из A в B, количество маршрутов из вершины 1 в вершину B увеличивается на $WAYS[A]$. Так как мы добавляем рёбра в порядке возрастания, гарантируется, что все эти маршруты — полезные. После добавления в граф всех ребёр в $WAYS[1]$ мы будем иметь количество полезных маршрутов из вершины 1 в саму себя. Одним из них будет и пустой маршрут, поэтому нужно вычесть 1 из этого количества.

Теперь рассмотрим, что надо делать в случае ребёр с одинаковым весом. Очевидно, что их добавление нужно произвести одновременно. Сделать это быстро можно с помощью вспомогательного массива.

Задача J. Сегменты

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дано N отрезков. Длина каждого из них, измеренная в сантиметрах — некое целое число. Требуется расположить эти отрезки на прямой таким образом, чтобы были выполнены следующие условия:

- Расстояния между левыми концами отрезков в сантиметрах также выражаются целыми числами.
- На прямой невозможно взять отрезок длиной L, в который полностью попадают хотя бы два данных отрезка (отрезок $[a, b]$ полностью попадает в отрезок $[c, d]$, если $c \leq a$ и $b \leq d$).
- Среди всех концов отрезков, расстояние между самым левым и самым правым - наименьшее возможное среди всех расположений отрезков, удовлетворяющих предыдущим двум пунктам.

Найдите, какое наименьшее расстояние, опять таки в сантиметрах, может быть между самым левым и самым правым концами отрезков.

Ограничения

$1 \leq N \leq 50000$

$1 \leq L \leq 1,000,000,000$

Длина каждого из отрезков находится в диапазоне $\left[1, 10^9\right]$.

Формат входного файла

Первая строка содержит целые числа N и M. Каждая из следующих N строк содержит длину отрезка.

Формат выходного файла

Единственное целое число — минимальное возможное расстояние между концами отрезков.

Пример

j.in	j.out
4 3 2 1 2 5	6
3 2006 2006 2006 2006	2008

Разбор задачи J. Сегменты

Естественно, что отрезки длиной более L можно позиционировать как угодно. Поэтому найдём длину наибольшего из таких отрезков и не будем их более рассматривать. Теперь перейдём к отрезкам длиной L и меньше. Допустим, мы выбрали последовательность, в которой будем выкладывать отрезки на прямую, и длины отрезков в этой последовательности $L[1], L[2], \ldots, L[N]$. Выкладывать будем слева направо. Первые два отрезка надо выложить так, чтобы расстояние между их наиболее удалёнными концами стало *L+1*. При выкладке третьего отрезка расстояние между наиболее удалёнными концами второго и третьего отрезков также должно быть *L+1*, соответственно расстояние от левого конца первого отрезка до правого конца третьего будет $2L + 2 - L[2]$. Следуя этой логике, мы получим, что после выкладки последнего отрезка расстояние между левом концом первого и правым концом N-ого отрезков выражается формулой

$$\mathbf{S} = N * (L + 1) - SUM(\mathbf{L}[1], \ldots, \mathbf{L}[N]) + \mathbf{L}[1] + \mathbf{L}[N]$$

Все слагаемые в этой формуле, кроме величин $L[1]$ и $L[N]$ — постоянные, соответственно, для минимизации S требуется лишь минимизация $L[1]$ и $L[N]$. Следовательно, первым и последним отрезком последовательности должны быть взяты отрезки с наименьшими длинами, а порядок остальных отезков значения не имеет. Ответ задачи — максимум между S и длиной наибольшего отрезка.

Задача K. Шоколадная Фабрика

Имя входного файла:	`k.in`
Имя выходного файла:	`k.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вилли Вонка владеет шоколадной фабрикой, которая производит множество разных сладостей. Иногда Вилли Вонка раздаёт сладости бесплатно. Перед фабрикой выстраивается длиннющий ряд детишек и каждому что-нибудь вручают.

Скоро наступит очередной из таких довольно редких случаев. На этот раз было решено раздавать конфеты. Фабрика производит N различных видов конфет. Вилли Вонка собирается отдать каждому из детей набор из $N-1$ штук различных конфет. Беда только в том, что количества конфет разных видов могут различаться, и становится трудно подсчитать, скольким детям достанется подарок при такой схеме в наилучшем случае. Вот это вам и придётся сделать.

Ограничения

$2 \leq N \leq 10000$

Количество конфет одного типа будет в диапазоне от 1 до 500,000,000, включительно.

Формат входного файла

Первая строка содержит число N - количество различных видов конфет. i-ая из следующих N строк содержит одно число - количество конфет i-ого типа.

Формат выходного файла

Единственное число - максимальное количество комплектов из $N-1$ конфет, которое можно получить, если распределять конфеты оптимально.

Пример

k.in	k.out
3 10 13 4	13

Разбор задачи K. Шоколадная Фабрика

Самым простым в реализации решением, на мой взгляд, является перебор двоичным поиском ответа задачи. Естественно, что если ответ задачи X, то можно получить любое количество комплектов между 0 и X, и нельзя получить количество $X+1$, $X+2$ и так далее. Поэтому двоичный поиск можно применить. Проверить же, возможно ли получить некоторое количество комплектов K, можно следующим образом.

Зафиксируем какой-либо один тип конфет как вспомогательный. Для всех остальных подсчитаем, в скольких из K комплектов он может участвовать: если количество конфет какого-либо типа Q меньше K, конфеты этого типа можно использовать только в Q комплектах, в противном же случае их хватит на все K комплектов. Все типы конфет, которых не хватает на все K комплектов, должны быть заполнены тем типом конфет, который был избран вспомогательным. Более того, в одном комплекте должно быть не более одного такого замещения. Соответственно, должно выполняться два условия:

- Общее количество недостающих конфет по всем типам, кроме вспомогательного, не должно превосходить количество конфет этого самого вспомогательного типа;
- Общее количество недостающих конфет по всем типам, кроме вспомогательного, не должно превосходить K.

Чтобы лучше понять логику, рассмотрим пример с четырьмя типами конфет. Есть всего 4 различных варианта комплекта: (1,2,3), (1,2,4), (1,3,4) и (2,3,4). Скажем, оптимальный вариант – выдать A штук комплекта (1,2,3), B штук комплекта (1,2,4), C штук комплекта (1,3,4) и D штук комплекта (2,3,4). Тогда $A+B+C$ не должно превосходить общее количество конфет типа 1.

Задача L. Строки

Имя входного файла:	`l.in`
Имя выходного файла:	`l.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дана строка, состоящая только из букв обоих регистров латинского алфавита. Над строкой неограниченное количество раз могут быть выполнены следующие действия:

- выделение всей строки;
- взаимозамена правой и левой половин выделенной части, если она четной длины;
- взаимозамена левой от центрального символа и правой от центрального символа частей выделенной части, если она нечетной длины;
- выделение только правой половины выделенной части, если она четной длины;
- выделение только левой половины выделенной части, если она четной длины;
- выделение только правой от центрального символа части выделенной части, если она нечетной длины;
- выделение только левой от центрального символа части, выделенной части, если она нечетной длины;
- выполнение любого из перечисленных действий над выделенной частью строки.

Очевидно, центральный символ выделенной части, если ее длина нечетная, в дальнейшем не меняется. Также очевидно, что если выделенная часть состоит из одинаковых символов, то в дальнейшем и она не меняется.

Например, строку “baCa” можно преобразовать следующим образом:

$$“baCa'' \rightarrow “Caba'' \rightarrow “Caab'',$$

а также и так:

$$“baCa'' \rightarrow “baaC'' \ldots$$

А строку “dbdCd” можно преобразовать так:

$$“dbdCd'' \rightarrow “bddCd'' \rightarrow “Cddbd'' \rightarrow “Cdddb''$$

Наша задача — для заданных строки и символа определить длину максимальной непрерывной подстроки, состоящей только из заданного символа, которую можно получить путем преобразования заданной строки вышеуказанным способом.

Ограничения

Длина заданной строки не меньше, чем 1 и не больше, чем 100,000.

Формат входного файла

Входной файл содержит две строки. В первой строке задан нужный символ, а во второй - исходная строка.

Формат выходного файла

Выходной файл содержит единственное число — ответ задачи.

Пример

l.in	l.out
a abCa	2
d dbdCd	3

Разбор задачи L. Строки

Для данных строки S и символа C определим следующие функции:

- **Left***(S,C)* — обозначает, какое максимальное количество символов C можно собрать в начале строке S в результате операций.
- **Right***(S,C)* — обозначает, какое максимальное количество символов C можно собрать в конце строки S в результате операций.
- **Sol***(S,C)* — максимальное количество символов C, которое можно собрать в строке S, т.е. ответ задачи.

Если длина S — один символ, то:

- Если $S[0] == C$, **Left**(S,C)=**Right**(S,C)=**Sol**(S,C)=1.
- Если $S[0] != C$, **Left**(S,C)=**Right**(S,C)=**Sol**(S,C)=0.

Рассмотрим теперь, что происходит, когда строка S чётной длины. Обозначим через L длину строки S, через A левую половину S, а через B - правую. Значением *Left(S,C)* будет максимум между следующими величинами:

- Если **Left**(A,C)==L/2 (т.е. вся строка A состоит из символов C), то **Left**(A,C)+**Left**(B,C). В противном случае, **Left**(A,C).
- Если **Left**(B,C)==L/2 (т.е. вся строка B состоит из символов C), то **Left**(B,C)+**Left**(A,C). В противном случае, **Left**(B,C).

Аналогичное выражение получим и для *Right(S,C)*. Значение *Sol(S,C)* — максимум между величинами:

- **Left**(S, C).
- **Right**(S, C).
- **Right**(A, C)+**Left**(B, C).
- **Right**(B, C)+**Left**(A, C)).

В случае, когда длина строки нечётна, получаются схожие формулы с учётом символа в центре строки.

Задача M. Стрекоза и муравей 2

Имя входного файла:	`m.in`
Имя выходного файла:	`m.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Сложные и многогранные взаимоотношения между стрекозой и муравьем отражены отнюдь не только в известной басне Крылова. В частности, согласно одноименной грузинской народной сказке, стрекоза выручает муравья, попавшего в реку до того, как тот успел научиться плавать. Кстати, о том как обстоят дела у муравья в этом смысле в данный момент, народный эпос пока умалчивает. Рассказ о том, как все же стрекоза сумела выручить муравья из беды, заслуживает отдельной задачи. Мы же проследим за событиями с того момента, когда спасенный муравей решил пригласить своего спасителя в лучшую хинкальную леса, в котором они обитали. В момент, когда муравью пришла в голову такая мысль, друзья находились на краю опушки, которую можно представить как прямоугольный участок размером MxN. Каждая клетка этого участка характеризуется своей высотой от уровня моря. При этом, друзья находятся в клетке (1,1), а вожделенная хинкальная расположена на другом конце опушки в клетке (M,N). Каждым очередным шагом друзья могут попасть в любую из соседних клеток, расположенных внутри опушки. Две квадратные клетки считаются соседними, если у них есть общяя сторона.

Перед друзьями стала задача подобрать такой маршрут, чтобы возможно меньше устать. Для стрекозы предпочтителен маршрут, проходящий по меньшему количеству клеток, а для муравья — маршрут, в котором минимальна максимальная высота, которую ему придется взять. Начальная и конечная клетки входят в маршрут. Так как по версии грузинской народной сказки и стрекоза, и муравей хорошо воспитаны, то каждый из них настаивает на варианте, объективно предпочтительном для друга. Но учитывая,

что это муравей пригласил стрекозу, а не наоборот, он настоял, чтобы был избран маршрут, наиболее удобный для стрекозы. В случае существования нескольких таких маршрутов будет взят тот из них, который устраивает муравья как можно больше.

Ограничения

M и N — целые числа, причем $2 \leq M, N \leq 500$. Высота каждой клетки — это целое число, принадлежащее диапазону 1..1,000,000,000.

Формат входного файла

Первая строка содержит значения чисел M и N. Каждая из следующих M строк содержит по N чисел. j-ое число $i+1$ - ой строки соответствует высоте клетки (i,j).

Формат выходного файла

Единственная строка с парой целых чисел — длина и максимальная высота маршрута, соответственно.

Пример

m.in	m.out
4 5 2 3 2 1 2 1 4 1 4 2 2 1 2 3 2 1 2 1 4 1	8 3

Пояснение

Стрекозу устраивает, например, маршрут (1,1)–(1,2)–(1,3)–(1,4)–(1,5)–(2,5)–(3,5)–(4,5), в котором учтены и интересы муравья.

Разбор задачи M. Стрекоза и муравей 2

См. разбор к первой части задачи C — Tale.

Задача N. Нелюбимые цифры 2

Имя входного файла:	`n.in`
Имя выходного файла:	`n.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Новый руководитель организации обнаружил, что его предшественник, по одному ему известным причинам, для нумерации официальных документов принципиально не использовал числа, десятиричные записи которых содержали некоторые цифры. Причем в разные годы обструкции подвергались разные комплекты цифр. В качестве начального номера в каждом году предыдущий руководитель брал минимальное неотрицательное число, не содержашее отвергнутых в данном году им цифр. При нумерации каждого следующего документа, в тех случаях, когда следующий номер содержал отвергнутую цифру, он просто пропускал это число. И так до тех пор, пока очередное число не оказывалось свободным от нежелательных ему цифр. Например, если отвергались 8, 7, 9, 5, 1, то первые несколько документов этого года имели следующие номера: 0, 2, 3, 4, 6, 20, 22, 23, 24, 26, 30, 32, 33,. . .

И так как предшественник руководил организацией довольно долго и накопилось изрядное количество перенумерованных им документов, у нового руководителя возникла потребность в программе, которая для заданного комплекта отвергнутых цифр по исходному номеру документа (т.е. по номеру, который был ему присвоен) быстро определит порядковый номер этого документа, отсчитанный с нуля.

Ограничения

Общее количество отвергнутых цифр не менее 1 и не более, чем 8.

Исходный номер искомого документа не меньше нуля и не превосходит 1,000,000,000.

Формат входного файла

Входной файл содержит две непустые строки. В первой строке через пробел перечислены нелюбимые цифры (их общее количество от одной до восьми включительно). Во второй строке дан исходный номер искомого документа.

Формат выходного файла

Выходной файл содержит единственное число — номер запрошенного

документа, т.е. порядковый номер документа, исходный номер которого указан во входном файле.

Пример

n.in	n.out
8 7 9 5 1 24	8

Разбор задачи N. Нелюбимые цифры 2

Вначале замечаем, что нас интересуют цифры, которые разрешено использовать (назовем их допускаемыми).

Если среди этих цифр есть 0, то мы имеем дело с позиционной системой счисления, основание которой равно количеству допускаемых цифр. В этой системе в качестве цифр используются символы, не обязательно совпадающие по написанию с символами, используемыми в качестве цифр в стандартной позиционной системе счисления с таким же основанием. Однако имеется возможность установить взаимно однозначное соответствие с числами такой системы Например, если допускаются 3, 7, 6 и 0, то каждому номеру, проставленному предыдущим начальником, однозначно соответствует число в 4-ичной системе, которое получается из исходного заменой 3 на 1, 6 на 2 и 7 на 3. А так как числа стандартной 4-ричной системы составляют последова-тельность, заполняющую множество натуральных чисел полностью (т.е. без “дырок”), то остается лишь путем замены цифр перевести исходный официальный номер в стандартную позиционную систему счисления с основанием, равным количеству допускаемых цифр (промежуточную систему), а затем из полученной записи стандартным путем восстановить ее численное значение.

Если же среди допускаемых цифр нет цифры 0, то перед нами система без нулей. В качестве промежуточной системы счисления будем использовать нестандартную (без нуля) систему счисления с основанием, равным количеству допускаемых цифр. Эта система использует цифры, начиная с 1 до цифры, равной основанию системы и покрывает все целые числа, начиная с 1. Например, если допускаемые цифры 3, 8 и 4, то соответствующей промежуточной системой будет 3-ичная система с цифрами 1,2,3, в которой 1 соответствует 3, 2 соответствует 4, 3 соответствует 8. Перевод в такую систему осуществляется обычной заменой цифр. Восстановление численного значения такой запииси также происходит стандарным методм. А так как нас интересует номер, отсчитанный от 0, то полученный таким образом результат следует уменьшить на 1. Например, рассмотрим процесс получения

порядкового номера документа с исходным номером “883”, полученного в условиях, когда допускаются цифры 3,4,8. В начале заменой цифр переводим этот номер в промежуточную систему с цифрами 1,2,3. Получаем “331”. Затем получаем численное значение этой записи $1+3*3+3*3^2 = 37$, после чего вычитая из полученного результата 1, получаем окончательный ответ 36.

Задача O. Снова палиндромы

Имя входного файла:	`o.in`
Имя выходного файла:	`o.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Бека заинтересовался свойствами последовательности, которая получается выписыванием подряд цифр десятиричных представлений заданной последовательности неотрицательных целых чисел. Застав Беку за выписыванием вышеупомянутых цифр, Анна обратила внимание на участки полученной цепочки чисел, представляющие собой палиндромы (палиндромом, как известно, называется симметричная последовательность символов, т.е. такая последовательность, в которой равны попарно все символы, равноудаленные от концов последовательности).

Дети задались целью обнаружить палиндром максимальной длины. Давайте поможем им. Составим программу, которая по заданной последовательности неотрицательных чисел выдаст длину максимального палиндрома, являющегося непрерывной подпоследовательностью последовательности, полученной из цифр десятиричных значений исходной последовательности, выписанных подряд. Например, если задана последовательность 7, 12, 43, 421, 7503, то последовательность цифр этих чисел будет 7, 1, 2, 4, 3, 4, 2, 1, 7, 5, 0, 3. Очевидно, первые 9 членов этой последовательности составляют палиндром, который является самым длинным. Соответственно, нашим ответом будет 9.

Ограничения

Количество чисел, цифры которых будут служить строкой для нахождения палиндрома, не превышает 500. Каждое из этих чисел не менее 0 и не более 2,000,000,000.

Формат входного файла

Входной файл содержит числа, составляющие исходную последовательность. Числа разделены пробелами и/или переводами строки.

Формат выходного файла

Выходной файл содержит единственное число — ответ задачи.

Пример

o.in	o.out
332 7 12 43 421 750 233	9

Разбор задачи O. Снова палиндромы

Для начала получим из заданной последовательности чисел строку. Самый простой способ это сделать — прочесть входящий файл посимвольно и каждый символ, являющийся цифрой, добавить в конец строки. Учитывая, что все числа содержат не более 10 знаков, строка в итоге будет длиной не более 5000.

Теперь попытаемся найти самую длинную подстроку полученной строки, являющуюся палиндромом. Всего подстрок у строки длиной L есть *L*(L+1)/2* штук, а проверка каждой из них может требовать до *L/2* сравнений - соответственно, проверка всех подстрок данной строки даёт нам алгоритм сложности $O(L^3)$, что слишком медленно для данных ограничений.

Обозначим через *S(i, j)* подстроку строки *S*, начинающуюся с *i*-ого символа и заканчивающуюся *j*-ым. Если *S(i, j)* является палиндромом, то, очевидно, палиндромом является и *S(i+1, j-1)*, и *S(i+2, j-2)*, и так далее. Следовательно, для каждого символа k можно определить за линейное время, серединой палиндрома какой максимальной длины он является: проверять равенство символов справа и слева от *k*-ого до тех пор, пока символы не окажутся различны или не будет достигнут один из концов строки. Учитывая, что палиндромы могут быть и чётной длины, нужно также проверить для каждой пары соседних символов, серединой какого максимального палиндрома чётной длины они являются. Следовательно, для строки длиной L можно таким образом найти надлиннейший палиндром за $O(L^2)$.

Задача P. Преобразование

Имя входного файла:	`p.in`
Имя выходного файла:	`p.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Как известно, для облегчения автоматизации обработки алгебраических выражений используются бесскобочные способы их записи, называемые польскими в честь их автора, польского математика Лукасевича. Различают два вида польских записей - постфиксный (когда знак операции записывается непосредственно после опреандов) и префиксный (когда знак операции записывается непосредственно перед соответствующими операндами). Например, выражению *(a+b)*c-d/(e-f)* соответствует постфиксная запись *ab+c*def-/-* и префиксная запись *-*+abc/d-ef*.

В первой записи *ab+* сответствует *a+b*, далее рассматривая *ab+* как операнд, замечаем, что *ab+c** соответствует *(a+b)*c*. Далее *ef-* соответствует *e-f*, дальше *def-/* соответствует *d/(e-f)* и в итоге *ab+c*def-/-* соответствует *(a+b)*c-d/(e-f)*. Аналогично можно убедиться в том, что приведенная префиксная запись соответствует исходному выражению.

Подробный рассказ о пользе бесскобочных записей явно выведет нас из рамок регламента, принятого для условий задач олимпиад по программированию. Однако, все же отметим, что постфиксная форма записи существенно облегчает вычисление значения выражения, а префиксная форма также существенно облегчает автоматизацию функциональных преобразований выражения. И так как существуют задачи, в которых возникает необходимость и в вычислении выражения, и в осуществлении его функциональных преобразований, актуальна задача преобразования выражения из одной бесскобочной формы в другую.

Вам вменяется написать программу, которая по заданному строкой бесскобочному выражению выдаст строку с другой бесскобочной формой того же выражения (если задана постфиксная запись, мы должны получить соответствующую префиксную и наоборот, если задана префиксная запись, необходимо получить соответствующую постфиксную запись).

Для простоты нам будет задаваться запись, соответствующяя выражению, в котором в качестве знаков операций используются: + - * / . Причем все операции только двуместные. В качестве операндов могут быть использованы только десятеричные цифры от 1 до 9, либо переменные, обозначенные односимвольным идентификатором, который может представлять собой только букву малого регистра латинского алфавита.

Ограничения

Длина заданной строки не меньше, чем 1 и не больше, чем 1000. Строка не содержит пробелов.

Формат входного файла

Входной файл содержит единственную строку — одну из бесскобочных форм исходного выражения, удовлетворяющую условиям задачи.

Формат выходного файла

В выходной файл следует выдать единственную строку — префиксную форму исходного выражения, если оно было задано в постфиксной форме, или постфиксную форму исходного выражения, если оно было задано в префиксной форме.

Пример

p.in	p.out
5	5
a9+c*d5f-/-	-*+a9c/d-5f
-*+abc/d-ef	ab+c*def-/-

Разбор задачи P. Преобразование

Акцепт по этой задаче, по сути, вознаграждение для тех, кто был внимателен на лекции.

Задача Q. Равносильность

Имя входного файла:	`q.in`
Имя выходного файла:	`q.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Школьник Бека обнаружил, что одно и то же сотношение можно записать по-разному. Например: “$x \geq 25$” можно записать и как “$x - 25 \geq 0$”, и как “$25 \leq x$” и даже как “$2 * x - 3 * y - 12 \geq x - y * 3 + 13$”. Он стал проверять равносильность различных соотношений, приведенных в одном толстом задачнике по математике. Напомним, что два соотношения называются равносильными, если для любого комплекта значений переменных, использованных в них, либо оба эти соотношения верны, либо оба они не верны (т.е. если их т.н. истинностные значения равны всегда).

Задавшись целью проверить эквивалентность абсолютно всех соотношений из книги, Бека вскоре понял, что ему необходима программа, которая поможет ускорить эту работу. Программировать он еще не умеет, поэтому просит о помощи Вас.

Нужно составить программу, которая по заданным двум соотношениям выдаст заключение об их равносильности на английском языке – либо YES, либо NO. Каждое из соотношений задано в виде строки. В качестве знака соотношения может быть один из следующих: < <= > >= = <> . В качестве операндов выражений, составляющих левую и правую части соотношения, могут быть либо двухсимвольные сочетания X0, X1,. . . , X9, обозначающие переменные, либо не более чем двузначные десятиричные числа (возможно с ведущим нулем). В качестве знаков операций в выражениях могут быть использованы следующие: – + *.

Ограничения

Длина каждой строки, содержащей исследуемое соотношение, не меньше 3 и не больше 1000. В последовательности, составленной только из знаков операций отдельно для левой и отдельно для правой частей, знак умножения не может встречаться дважды подряд. Строки не содержат пробелов. В выражении переменная может умножаться только на число. Аналогично, число может умножаться только на переменную. Тесты гарантируют, что соотношения заданы корректно. Т.е. у каждого соотношения есть ровно один знак сравнения, а также есть и правая и левая части, заданные в соответствии с условием.

Формат входного файла

Входной файл содержит две строки, удовлетворяющие приведенным ограничениям. В каждой строке по одному соотношению, удовлетворяющему приведенным условиям.

Формат выходного файла

Выходной файл содержит единственную строку. В этой строке с первой позиции должен быть текст YES, если строки эквивалентны, а в противном случае должен быть (также с первой позиции) текст NO.

Пример

q.in	q.out
2<-1 X0+32=0*X1-X0-25+X0*2	YES

Разбор задачи Q. Равносильность

Вначале замечаем, что правая и левая части парссматриваемых соотношений могут быть только линейными выражениями, содержащими не более чем 10 переменных. Учитывая еще и свободный член, каждую из частей соотношения можно взаимно однозначно отобразить на вектор, состоящий из 11 целых чисел. Очевидно, удобно выделить подстроку, содержащюю выражение, составляющее одну из частей соотношения и иметь подпрограмму, которая переводит строку в соответствующий вектор. Дальше, вычитая почленно из одного вектора другой, получаем эквивалент заданного выражения с нулем в правой части. Добьёмся положительного знака у первого ненулевого элемента вектора (при необходимости изменим знак сравнения на противоположный). Получив подобное представление для обоих сравниваемых соотношений, останется рассмотреть 3 случая:

- Оба соотношения не содержат переменных. В этом случае просто необходимо сравнить их единственные значения.
- Хотя бы одно из соотношений содержит переменную, но знаки сравнения у них разные. В этом случае, ответ отрицательный.
- Сравниваем почленно векторы представлений. Если встретился нуль, которому соответствует не нуль, либо соответствующие элементы имеют разные знаки, то ответ отрицательный. Осталось предусмотреть возможность разных векторов, соответствующие элементы, которых пропорциональны.

Задача R. Однажды в Китае 2

Имя входного файла:	`r.in`
Имя выходного файла:	`r.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Скоро увидит свет новая компьютерная игра “Однажды в Китае”. Пока же приходится довольствоваться демоверсией. Игра заключается примерно в следующем: один джигит овладевает исскуством кунг-фу и начинает обходить разные бандитские логова, где он избивает плохишей и попутно забирает у них награбленные деньги. Уровень владения кунг-фу выражается неким неотрицательным целым числом. Всего в игре N бандитских логовов. i-тое логово имеет три показателя: Q_i, S_i и M_i. Число Q_i указывает минимальное владение кунг-фу, которое должно быть у джигита, чтобы он смог ограбить бандитов в логове i. Если его владение кунг-фу менее

Q_i, ему туда лучше не соваться. S_i - это сумма (в юанях), которую сможет забрать джигит из i-ого логова, если его владение кунг-фу Q_i.

Если же его кунг-фу лучше, то он может выбить у бандитов и побольше деньжат. А конкретно, если владение кунг-фу (обозначим его K) в диапазоне от Q_i до $Q_i * M_i$, джигит заберёт у бандитов сумму $S_i * \frac{K}{Q_i}$. Если владение кунг-фу превосходит $Q_i * M_i$, джигит заберёт ровно $S_i * M_i$, потому что больше оттуда забирать нечего.

Звучит всё просто, но есть один подвох. Кунг-фу джигит должен обучаться у мастера Цзен, который за каждый час тренировки берёт 1 юань. А как известно, чем выше уровень ученика, тем дольше ему на-до заниматься для повышения квалификации. Чтобы повысить уровень от X_1 до X_2 , парню нужно $A * (X_2^2 - X_1^2)$ часов.

Мастер Цзен проводит тренировки только длительностью в целое количество часов. Он может обучать в кредит, чтобы джигит вернул ему деньги после применения своих навыков.

Учитывая, что в начале игры джигит не владеет кунг-фу (т.е. его уровень 0) и его баланс на нуле, найдите максимально возможную прибыль (т.е. разницу между отбитыми у бандитов средствами и потраченной на тренировки суммой), которую он может получить

Ограничения

$1 \leq N \leq 1000$
$0 \leq A \leq 10$, число дано не более чем с 3 знаками после запятой.
$1 \leq Q, S \leq 1000$
$1 \leq M \leq 10$
Числа M_i, S_i и Q_i — целые.

Формат входного файла

Первая строка содержит числа N и A.

Каждая из следующих N строк содержит числа Q_i, S_i и M_i.

Формат выходного файла

Одно число — максимальная прибыль джигита. Ваш ответ должен отличаться от правильного не более, чем на 0.000001 (1e-6).

Пример

r.in	r.out
1 0.15 5 10 3	6.65

Разбор задачи R. Однажды в Китае 2

Также, как и в усложнённой версии этой задачи, справедливы следующие наблюдения:

a) джигит может сначала доучиться до определенного уровня, а уже затем пойти добывать деньги, больше не проводя тренировок;

b) герою всегда выгодно ограбить все логова, которые он сможет при своём уровне владения кунг-фу.

При данных в задаче ограничениях самым простым способом решения будет проверить величину прибыли джигита для всех показателей владения кунг-фу в пределах от 0 до 10000, включительно. Совершать проверку для больших значений владения кунг-фу не имеет смысла, так как величина $Q_i * M_i$ не может превосходить 10000 и, соответственно, увеличить выручку уже не удастся. Сама проверка тривиальна – пройти циклом по всем логовам и оценить, сколько можно взять из каждого, а затем вычесть из прибыли деньги, затраченные на тренировки.