

Министерство образования и науки, молодежи и спорта
Украины



Яndex



<ерат>



miratech

Зимняя школа по программированию

Харьков, ХНУРЭ

2013

Оглавление

День первый. Контест Akai	8
Об авторе...	8
Теоретический материал. Топологическая задача о маршрутах.	9
Задачи и разборы	13
Задача A. Automaton	13
Задача B. Bims	15
Задача C. Cutting	17
Задача D. Disclosure	19
Задача E. Embedded circles	20
Задача F. False figures	22
Задача G. Grouping	24
Задача H. Hidden triangles	26
Задача I. Interactive	28
Задача J. Journey	30
Задача K. Knuth knows	31
Задача L. Lake	33
Задача M. Match them up	34
Задача N. Need for sum thing	36
Задача O. Open air	38
Задача P. Pseudo automaton	40
Задача Q. Quiz	41
Задача R. Reduction	44
День второй. Контест Бондаренко Натальи Павловны	45
Об авторе...	45
Теоретический материал. Поток в сетях	45
Задачи и разборы	50
Задача A. Самая простая задача	50
Задача B. Оптимальный поток величины 1	51
Задача C. Максимальный поток в неориентированном графе	52
Задача D. Задача про минералку	53
Задача E. В поисках невест	55
Задача F. Задача о назначениях	57
Задача G. План эвакуации	58
Задача H. Два кратчайших пути	61
Задача I. Остоз	62
Задача J. Поток в меняющейся сети	63
Задача K. Произвольная циркуляция	65
Задача L. Сетевые войны	66
Задача M. Снег в Берляндии	69

День третий. Контест Алексея Шмелева	72
Об авторе...	72
Теоретический материал. Использование STL для решения олимпиадных задач (алгоритмы, контейнеры, примеры, ошибки)	72
Задачи и разборы	84
Задача A. DOMA 2: Last Hit	84
Задача B. Last Effect 3: Danger	86
Задача C. Fineage: Training	88
Задача D. DOMA 2: Inventory	89
Задача E. South Mark: 3.50	92
Задача F. HOLM 2: The Great Battle	94
Задача G. Failout Few Vegas: Slow Save	97
Задача H. Carmarandom TDC2013: New Trace	99
Задача I. X-Dom: Railway	101
Задача J. MindCraft: Heliport	103
Задача K. Double Life: Amplifiers	105
День четвертый. Контест Неспирного Виталия Николаевича	108
Об авторе...	108
Теоретический материал. Линейное программирование и матричные игры	109
Задачи и разборы	115
Задача A. Неприводимые многочлены High	115
Задача B. Неприводимые многочлены Junior	116
Задача C. До первого выпадения High	121
Задача D. До первого выпадения Junior	122
Задача E. Раскраска забора High	125
Задача F. Раскраска забора Junior	126
Задача G. Города-побратимы High	128
Задача H. Города-побратимы Junior	129
Задача I. Отдых у реки High	131
Задача J. Отдых у реки Junior	133
Задача K. Количество представимых High	136
Задача L. Количество представимых Junior	136
Задача M. Квадратичная перестановка High	138
Задача N. Квадратичная перестановка Junior	138
Задача O. Цикл де Брёйна High	140
Задача P. Цикл де Брёйна Junior	141
Задача Q. Карточный поединок High	142
Задача R. Карточный поединок Junior	143
Задача S. Посадка в самолет High	146
Задача T. Посадка в самолет Junior	147

День пятый. Контест Геральда Агапова и Фефера Ивана	149
Об авторах...	149
Теоретический материал. Топологическая сортировка онлайн	150
Задачи и разборы	150
Задача А. Обнаружение циклов	150
Задача В. Топологическая сортировка	152
Задача С. Расписание на дереве	154
Задача D. Братья по крови наносят ответный удар	157
Задача Е. Гиперраздление	159
День шестой. Контест Фефера Ивана и Геральда Агапова	162
Теоретический материал. Суффиксное дерево	162
Задачи и разборы	166
Задача А. Черно-белый куб	166
Задача В. Один-два	168
Задача С. Оптимальное разрезание	169
Задача D. Граф-турнир	171
Задача Е. Замок в виде мини-игры	173
Задача F. Количество запросов	176
Задача G. Две перестановки	179
Задача H. Робот-конь	181
Задача I. Прыг-скок	183
Задача J. Суффиксное дерево	185
Задача K. Суффиксное дерево двух строк	186
Задача L. Уникальные суффиксы	189
Задача M. Общая подстрока	191
Задача N. Контролирующее множество строк	192
День седьмой. Контест Куликова Егора Юрьевича	195
Об авторе...	195
Теоретический материал. Суффиксный автомат	195
Задачи и разборы	200
Задача А. Chords	200
Задача В. Cyclic suffixes	201
Задача С. A Coloring Game	203
Задача D. Hippopotamus	204
Задача Е. False RSA	205
Задача F. Perspective	206
Задача G. Circular Railway	208
Задача H. SETI	209
Задача I. 2-3 Trees	210
День восьмой. Контест Гольдштейна Виталия Борисовича	213
Об авторе...	213

Теоретический материал. Кратчайшие пути в графах	214
Задачи и разборы	216
Задача А. Красная Шапочка	216
Задача В. Уборка снега	219
Задача С. Юный поджигатель	221
Задача D. Реклама	229
Задача Е. Космический мусорщик	233
Задача F. Shortest Path	235
Задача G. Дом улыбок	236
Задача H. Сфера	238
Задача I. Домой на электричках	239
Задача J. Поле чудес	240
Задача K. Квадрат	244
Задача на самый короткий код	248
День восьмой. Контест Рипатти Артема Валерьевича	250
Об авторе...	250
Задачи и разборы	250
Задача А. Игра со стрелками	250
Задача В. Гарри Поттер и философский камень	253
Задача С. Преобразование последовательности	255
Задача D. Отрезки	257
Задача Е. Четыре подмножества	260
День девятый. Контест Копелиовича Сергея Владимировича	263
Об авторе...	263
Теоретический материал. Динамика и жадность 20 лет спустя	264
Задачи и разборы	280
Задача А. Белоснежка и n гномов	280
Задача В. Эльфы и олени	281
Задача С. Приключение	283
Задача D. Авторитеты	284
Задача Е. Коробки	285
Задача F. Простые пути в дереве	287
Задача G. Редукция дерева	288
Задача H. Изоморфные деревья	289
Задача I. К минимумов на отрезке	291
Задача J. Инверсии отрезка	292
Задача K. Продуктивный бинпоиск	293
Задача L. Командный пункт	294
Задача M. Общая подпоследовательность	296
Задача N. Различные подпоследовательности	297
Задача O. Наибольшая общая возрастающая	298

Задача Р. k-Рюкзак	299
Задача Q. Разбиения на слагаемые	300
Задача R. Волшебный лес	301

День первый (16.02.2013 г.) Контест Akai

Об авторе...

Миланин Александр, родился 28 августа 1988 года. Окончил Таврический национальный университет им. В.И. Вернадского. Аспирант кафедры “Прикладной математики”. Ассистент кафедры “Информатики”.

Основные достижения:

В составе команды Akai занял:

- В 2009 году — 6 место на полуфинале чемпионата мира региона SEERC, Бухарест.
- В 2010 году — 3 место в финале чемпионата Украины в городе Винница.
- В 2010 году — 4 место в полуфинале чемпионата мира региона SEERC в Бухаресте.
- В 2011 году — 1 место во II этапе Всеукраинской студенческой олимпиады по программированию.
- В 2011 году — 2 место в полуфинале чемпионата мира региона SEERC в Бухаресте.
- В 2010 и 2012 годах — 1 место в регионе SEERC Открытого кубка им.Панкратьева.
- Финалист Russian Code Cup 2011 и 2012 годов.
- 3 место на Russian AI Cup 2012.



Теоретический материал. Топологическая задача о маршрутах.

Постановка задачи

Пусть есть прямоугольная карта размера $N \times M$, разбитая на квадратные клетки 1×1 . Некоторые из клеток являются занятыми, остальные свободные. На этой карте имеются два циклических ориентированных маршрута. Оба маршрута проходят через пустые клетки таким образом, что любые две соседние клетки в плане маршрута являются соседними в одном из четырех направлений: верх, низ, лево, право. То есть каждый из маршрутов можно описать координатами его начальной клетки и списком команд четырех типов: вверх (U), вниз (D), влево (L), вправо (R). Команды описывают движение по маршруту.

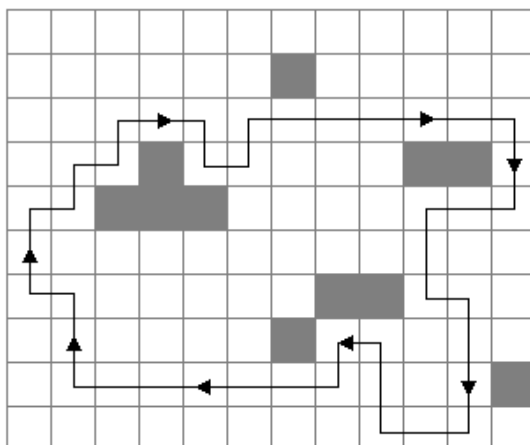
Требуется определить, являются ли эти маршруты топологически эквивалентными. То есть могут ли они быть получены друг из друга гомеоморфным преобразованием. Понятие гомеоморфизм вводится для пространств, обладающих свойством непрерывности, что противоречит дискретной постановке задачи.

Однако можно ввести некоторое эквивалентное понятие непрерывного преобразования маршрута на карте. А именно: пусть $S = a_1 a_2 a_3 \dots a_l$, $a_i \in \{L, R, U, D\}$ — список команд, соответствующий некоторому маршруту. Непрерывным преобразованием называется последовательность действий четырех типов:

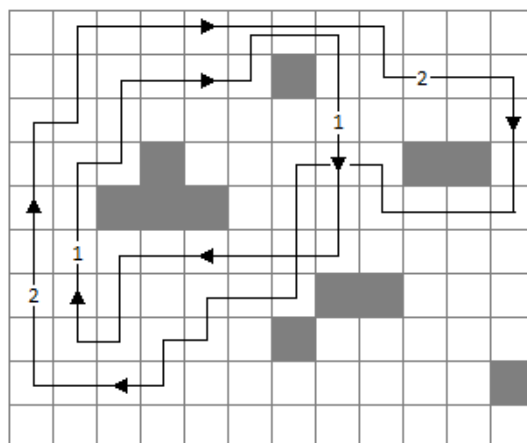
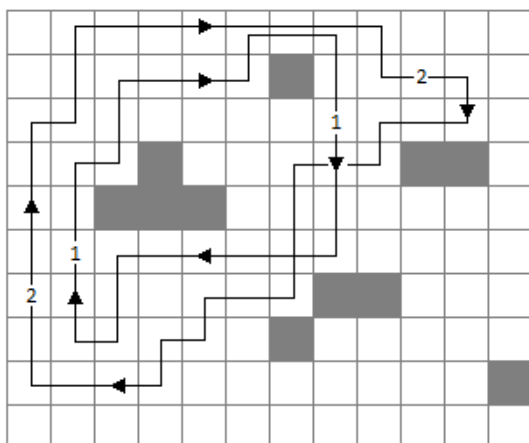
- Для некоторой позиции i , вставить пару подряд идущих взаимнообратных команд (LR, RL, UD, DU). То есть, $S' = a_1 a_2 \dots a_{i-1} L R a_i \dots a_l$.
- Обратное действие. Для некоторой позиции i удалить пару соседних взаимнообратных команд.
- Для некоторой позиции i , вставить команду перед элементом a_i и обратную к ней после элемента a_i . То есть, $S' = a_1 a_2 \dots a_{i-1} L a_i R a_{i+1} \dots a_l$.
- Обратное действие. Для некоторой позиции i удалить пару взаимнообратных команд на позициях $i - 1$ и $i + 1$.

Пояснение

На изображении ниже показана карта и некоторый маршрут на ней.



На изображении ниже показаны топологически эквивалентные маршруты (слева) и не эквивалентные (справа)



Если два маршрута одинаковые, но имеют разные направления, они не являются эквивалентными.

Решение

Предлагается каждый из маршрутов привести к некоторому каноническому виду. Каждый маршрут из одного класса эквивалентности сводится к одному каноническому виду. Очевидно, что набор фигур внутри маршрута не определяет класс эквивалентности хотя бы потому, что все эти фигуры мы могли бы обходить в другом направлении.

Основная идея — стянуть маршрут так, чтобы он имел минимальную длину. Представляется, что маршрут — это веревка, которая превращается в резинку и натягивается на препятствиях.

Пусть мы начинали из точки $(3, 2)$ и маршрут выглядел как *RRRRDDR UURDDDDL LLLULDDL UUUUU*. Тогда порядок пересечений лазером будет $[-4, -6, 6, 4, -4, -6, -9, -10, 10, -10, -11, 11, 10, 8, 5, 4]$. Положительные числа соответствуют пересечению соответствующего лазера снизу вверх, а отрицательные — сверху вниз.

Очевидно, что если мы пересекли некоторый лазер сначала в одном направлении, а потом сразу же в обратном, то мы можем стянуть наш путь таким образом, чтобы мы эти два пересечения не делали. Еще очевидно, что если есть два пересечения подряд с одним лазером, то они шли в противоположных направлениях.

Пока в списке пересечений есть хоть одна пара соседних пересечений с одним и тем же лазером, удалим ее из списка. Стоит отметить, что список циклический, поэтому первый и последний элементы тоже соседние. В нашем примере список получится $[-6, -9, 8, 5]$.

Как только таких пар больше нет, это значит, что мы почти нашли канонический вид маршрута. Поскольку мы могли начать маршрут в любой точке маршрута, пересечения тоже могли начаться с любого лазера. Поэтому для однозначности списка найдем его лексикографически минимальный сдвиг. Для нашего примера, это $[-9, 8, 5, -6]$.

Мы бы могли не учитывать направление пересечения лазером, чтобы составить сам список, но поскольку нам важно направление движения вдоль маршрута, надо учитывать и направление пересечений.

Для обоих маршрутов найдем их лексикографически минимальный канонический список пересечений лазером. Маршруты эквивалентны тогда и только тогда, когда эти два списка равны. Канонический список пересечений с лазерами задает класс эквивалентности маршрутов, поэтому таким подходом можно решать более сложные задачи, например, классификации или кластеризации маршрутов.

Отметим, что удаление пар соседних пересечений с одним лазером в явном виде делать не надо. Можно воспользоваться стеком при создании списка. Если очередное пересечение совпадает с пересечением на верхушке стека, надо удалить это пересечение из стека. Если не совпадает, то добавить его в стек. Важно не забыть, что список циклический и надо произвести удаления одинаковых пересечений в начале списка и в конце. Еще один тонкий момент, что если маршрут стянулся в точку, то есть его список пересечений оказался пуст, нужно учитывать его начальную точку. То есть, два маршрута с пустыми списками пересечений будут эквивалентными, если их начальные точки находятся в одной компоненте связности. Иначе они не будут эквивалентными.

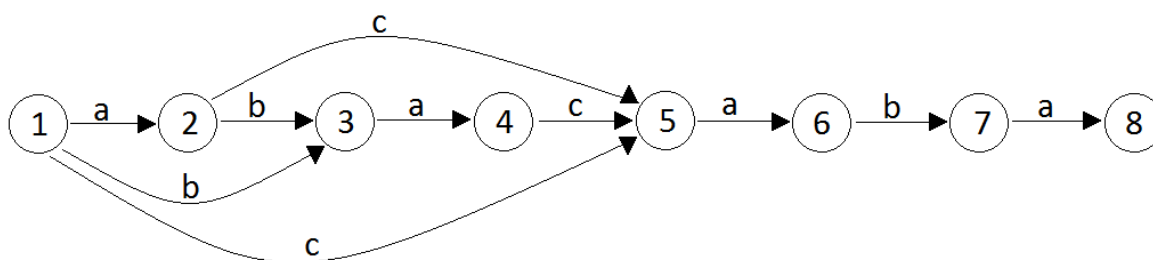
Задачи и разборы

Задача А. Automaton

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Дана строка S . Построить детерминированный конечный автомат, принимающий все суффиксы строки S (и возможно другие конечные строки). Автомат должен состоять из минимального числа состояний — N и не более чем $2N$ переходов. Каждое состояние автомата объявляется финальным. Начальное состояние имеет номер 1.

На изображении ниже показан автомат из тестового примера для строки “abacaba”



Формат входного файла

В единственной строке слово S , состоящее из строчных латинских букв.

Формат выходного файла

В первой строке два числа N и K — количество состояний и количество переходов. Далее K строк, в каждой из которых по два числа a_i , b_i и буква c_i , означающие наличие перехода из состояния a_i в b_i по букве c_i . Переходы можно выводить в любом порядке. Если решений несколько, можете вывести любое из них.

Ограничения

$$\begin{aligned}
 1 &\leq |S| \leq 10^5 \\
 1 &\leq K \leq 2N \\
 1 &\leq a_i, b_i \leq N \\
 'a' &\leq c_i \leq 'z'
 \end{aligned}$$

Пример

stdin	stdout
abacaba	8 10 1 2 a 1 3 b 1 5 c 2 3 b 2 5 c 3 4 a 4 5 c 5 6 a 6 7 b 7 8 a

Разбор задачи A. Automaton

Пусть N — длина строки S . Введем $N + 1$ состояние и N переходов, каждый из которых ведет из состояния i в состояние $i + 1$ по букве $S[i]$. Теперь конструктивно создадим не более N переходов, то есть, не более одного для каждого из суффиксов. Для этого будем в каждом состоянии хранить суффиксы, которые автомат должен принять, находясь в этом состоянии. Например, в первом состоянии будут все суффиксы от 2 до N .

Рассмотрим все состояния, начиная с 1 до N , для каждого из них будем делать следующее: список всех суффиксов, добавленных в это состояние, сгруппируем по первой букве, поскольку для каждой буквы из одного состояния может быть не более одного перехода. Рассмотрим каждую группу суффиксов с одинаковой первой буквой. Если по этой букве в автомате уже есть переход, (такой переход может быть в следующее состояние), то новый переход не создается, если перехода нет, выберем из группы самый длинный суффикс и сделаем переход в состояние, соответствующее этому суффиксу (номер состояния равен номеру суффикса плюс 1). Все суффиксы группы, сдвинутые на 1 вправо (так как мы делаем переход по первой букве), добавим в состояние автомата, в которое ведёт переход по первой букве суффиксов группы. Очевидно, что каждый добавленный переход переводит некоторый суффикс в такое состояние, из которого он примется без необходимости создания дополнительных переходов. То есть переходов будет добавлено не более $N - 1$. И одновременно, поскольку мы выбираем самый длинный суффикс для создания перехода, все остальные суффиксы переходят в новое состояние, сдвигаясь на 1, и размер этих суффиксов

меньше чем количество оставшихся состояний. То есть состояний хватит, чтобы принять все суффиксы.

Чтобы это работало быстро, заметим, что после добавления группы суффиксов в некоторое состояние, не имеет смысла больше добавлять суффиксы в это состояние (поскольку это будут те же самые, уже добавленные, суффиксы). А также, если изначально отсортировать все суффиксы лексикографически, то в процессе движения по состояниям, мы только разбиваем суффиксы на группы, не меняя их порядок. Поэтому для каждого состояния можно хранить два числа L и R , которые означают интервал суффиксов в суффиксном массиве, которые нужно принять из этого состояния, и число T — сдвиг каждого из них. Тогда для каждого состояния можно бинарным поиском разделять его суффиксы на группы по первой букве и с помощью дерева отрезков находить из них суффикс максимальной длины, чтобы определить, куда сделать переход по текущей букве. После этого из дерева отрезков как бы удаляется этот суффикс, чтобы он в дальнейшем не мешал выбирать самый длинный суффикс.

Итого, общая сложность получается $O(N \log(N))$ — построение суффиксного массива, бинарный поиск для разделения на группы и поиск минимума на интервале для каждого состояния.

Задача B. Bims

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

k -мерный симплекс Паскаля — это такая k -мерная фигура, что в ячейке (i_1, i_2, \dots, i_k) записано число $C_n^{i_1, i_2, \dots, i_k}$, где $i_j \geq 0$, а $n = i_1 + i_2 + \dots + i_k$.

Пример:

1-мерный симплекс Паскаля — это бесконечная строка из единиц

2-мерный симплекс Паскаля — это обычный треугольник Паскаля

3-мерный симплекс Паскаля — это бесконечный тетраэдр, в котором записаны сочетания вида $C_n^{i,j,k}$ и так далее.

Допустим, что у нас есть k -мерный симплекс Паскаля, в котором все сочетания записаны по модулю p (p — простое). Рассмотрим n -тый слой этого симплекса (тот, где записаны все сочетания из n). Необходимо найти количество ненулевых элементов на этом слое. Так как это число может оказаться большим, необходимо вывести его по модулю $10^9 + 7$.

Формат входного файла

В первой строке даны три числа k , p и t . В каждой из следующих t строк находится одно число n_i — номер слоя.

Формат выходного файла

Ровно t строк, в каждой из которых для соответствующего n_i вывести количество ненулевых элементов на n_i -том слое симплекса, по модулю $10^9 + 7$.

Ограничения

$$1 \leq k \leq 10^3$$

$$2 \leq p \leq 10^6 + 3, p \text{ — простое.}$$

$$1 \leq t \leq 10^5$$

$$0 \leq n_i \leq 10^{18}$$

Пример

stdin	stdout
2 7 4 0 6 7 8	1 7 2 4
3 7 4 0 6 7 8	1 28 3 9

Разбор задачи В. Vims

Рассмотрим первые p слоёв симплекса. Понятно, что нулей среди них не будет, так как все множители, входящие в сочетание, меньше p .

Рассмотрим p -тый слой. Там, напротив, почти все элементы будут нулевыми, если не считать элементы вида $C_p^{0,0,0,\dots,p,\dots,0,0,0}$, где p входит как в числитель, так и в знаменатель и сокращается.

Если расположить сочетания в симплексе так, что $C_n^{i_1,i_2,\dots,i_k}$ находится прямо под $C_n^{i_1-1,i_2,\dots,i_k}$, $C_n^{i_1,i_2-1,\dots,i_k}$, ... $C_n^{i_1,i_2,\dots,i_k-1}$ (нетрудно заметить, что оно будет равняться их сумме), то можно увидеть, что ненулевые элементы p -того слоя будут располагаться по углам этого слоя и генерировать ещё k симплексов высоты p .

Теперь видно, что если мы заменим каждый такой симплекс высоты p сочетанием, его порождающим, то получится точно такой же k -мерный симплекс Паскаля по модулю p . Отсюда возникает формула:

$$answer_n = answer_{n \bmod p} \cdot answer_{n/p}$$

Второй сомножитель можно искать рекурсивно. Первый же сомножитель равен просто количеству элементов на $(n \bmod p)$ -том слое и равен количеству способов разбиения $(n \bmod p)$ на k упорядоченных слагаемых, то есть $C_{(n \bmod p)+k-1}^{k-1}$.

Задача C. Cutting

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дана строка S и список M , состоящий из N слов. За одну операцию можно выбрать некоторую подстроку строки S , если такая встречается в качестве слова в списке M , и вырезать из строки S . После чего оставшиеся части строки S , если такие есть, склеиваются. Определить, за какое минимальное количество операций можно уничтожить всю строку S . Гарантируется, что это можно сделать.

Формат входного файла

В первой строке слово S . Во второй строке целое число N — количество слов в списке. Далее N строк, в каждой из которых по одному слову из списка M . Все слова состоят только из строчных латинских букв.

Формат выходного файла

Одно число — минимальное количество операций, необходимое для уничтожения строки S .

Ограничения

$$\begin{aligned} 1 &\leq |S| \leq 100 \\ 1 &\leq N \leq 100 \\ 1 &\leq |M_i| \leq 100 \end{aligned}$$

Пример

stdin	stdout
abacaba	3
4	
aba	
aca	
a	
b	

Пояснение

Первой операцией вырезали подстроку “aca”, получили “abba”.

Второй операцией вырезали подстроку “b”, получили “aba”.

Третьей операцией удалили всю строку “aba”.

Разбор задачи C. Cutting

Задача решается с помощью динамического программирования. Пусть $r[i][j]$ — минимальное количество удалений, необходимых, чтобы полностью удалить подстроку S от i до j . $d[i][j][k][l]$ — минимальное количество удалений, необходимых, чтобы от подстроки S от i до j остался префикс длины l слова k . Пересчитывается следующим образом:

$$d[i][j][k][l] = \min_{i \leq p \leq j} \{d[i][p-1][k][l-1] + r[p+1][j]\}$$

при условии, что $m[k][l] = S[p]$, где $m[k][l]$ — l -тая буква k -го слова из списка.

$$r[i][j] = \min \left\{ \min_{i \leq p \leq j-1} \{r[i][p] + r[p+1][j]\}, \min_{1 \leq k \leq N} \{d[i][j][k][len(k)]\} \right\}$$

где $len(k)$ — длина k -го слова.

Динамика работает за $O(|S|^3 \sum len(i))$ или проще говоря $O(N^5)$. Вычислительная сложность довольно высокая, но константа заметно меньше 1, и ее можно еще уменьшить, если не рассматривать недостижимые состояния. А именно, не рассматривать те сочетания i и $len(k)$, что $i + len(k) > |S|$. А также можно сократить используемую память до N^3 и даже N^2 .

Задача D. Disclosure

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 512 Мб

Дан ациклический ориентированный граф из N вершин и K ребер. Требуется удалить из него максимальное количество ребер, чтобы транзитивное замыкание графа не изменилось.

Транзитивное замыкание графа G — граф G' , состоящий из множества вершин исходного графа G и множества ребер (u, v) таких, что существует путь из вершины u в вершину v в графе G .

Формат входного файла

В первой строке два числа N и K . Далее K строк, в каждой из которых по два целых числа a_i и b_i , означающих наличие ребра, ведущего из вершины a_i в b_i . Граф не содержит петель, циклов и кратных ребер.

Формат выходного файла

Вывести те ребра, которые необходимо оставить в графе, в виде пар соединенных ими вершин. Ребра можно выводить в любом порядке.

Ограничения

$$1 \leq N \leq 50\,000$$

$$0 \leq K \leq 50\,000$$

$$1 \leq a_i, b_i \leq N$$

Пример

stdin	stdout
5 6	1 2
1 2	2 3
2 3	3 5
3 5	4 5
4 5	
1 5	
1 3	

Разбор задачи D. Disclosure

Ребро (u, v) можно удалить из графа, если существует путь из вершины u в вершину v . Любое другое ребро удалить нельзя.

Обойдем граф обходом в глубину. Для текущей вершины u будем определять $S[u]$ — множество всех вершин, до которых существует путь из вершины u . $S[u] = \bigcup S[v_i]$, где v_i — сын вершины u . Зная $S[u]$, можно удалить все ребра к детям, которые входят в это множество. После этого надо добавить в множество достижимых вершин всех детей текущей вершины.

Множества S надо хранить в виде битовой маски. Тогда объединение будет происходить за $N/32$ операций.

Общая сложность $O(NK/32)$

Задача E. Embedded circles

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дана таблица $\{a_{ij}\}$ из R строк и C столбцов, состоящая из цифр от '0' до '9'. Требуется ответить на Q запросов вида:

$i_k j_k r_k, 1 \leq k \leq Q$

Найти сумму всех элементов таблицы a_{ij} таких, что

$$(i - i_k)^2 + (j - j_k)^2 \leq r_k^2.$$

Область суммирования для каждого запроса представляет собой круг с центром в ячейке (i_k, j_k) и радиусом r_k . Область определяется в точности по формуле выше, но для удобства будем называть ее кругом.

Для любых двух запросов выполняется следующее: либо их круги не имеют общих ячеек таблицы, либо один из кругов полностью содержится в другом.

Более того, запросы идут в порядке вложенности. Это означает, что если круги запросов k и l ($k < l$) имеют общие ячейки, то из этого следует, что для любого t : $k < t \leq l$, круг t полностью содержится в круге запроса k . Круги различных запросов могут совпадать.

Формат входного файла

В первой строке два числа R и C — размеры таблицы. Далее R строк по C цифр в каждой (между цифрами нет пробелов). В следующей строке число Q — количество запросов. Далее в Q строках описаны запросы по

три целых числа: i_k , j_k , r_k , в каждой строке — номера строки и столбца центральной ячейки и радиус круга. Ячейки нумеруются с 1.

Формат выходного файла

Поскольку запросов много, выведите одно число — сумму всех ответов на запросы.

Ограничения

$$1 \leq R, C \leq 2\,000$$

$$'0' \leq a_{ij} \leq '9'$$

$$1 \leq Q \leq 10^6$$

$$1 + r_k \leq i_k \leq R - r_k$$

$$1 + r_k \leq j_k \leq C - r_k$$

$$0 \leq r_k \leq \frac{\min(R, C) - 1}{2}$$

Пример

stdin	stdout
6 6 123456 234567 345678 456789 567890 678901 10 1 1 0 3 3 2 3 2 1 2 2 0 4 2 0 1 3 0 2 3 0 4 3 0 5 5 1 5 5 0	141

Пояснение

$$141 = 1 + 65 + 20 + 3 + 5 + 3 + 4 + 6 + 25 + 9$$

Разбор задачи E. Embedded triangles

Каждый запрос можно обрабатывать за $O(r_i)$, где r_i — радиус круга запроса. Для этого можно посчитать все частичные суммы по каждой строке таблицы и разложить круг из запроса на строки. Для каждой строки круга за $O(1)$ найти ее сумму.

Докажем, что если запросы различны, то общая сложность будет $O(S)$, где S — площадь таблицы.

Пусть $f(R)$ — максимальное суммарное количество операций, необходимое, чтобы обработать все запросы в круговой области, при условии, что радиус максимального круга, который можно вписать в эту область, равен R .

$g(S)$ — максимальное суммарное количество операций, необходимое, чтобы обработать все запросы в произвольной области площади S .

$$f(R) = 2r + f(r - 1) + g(\pi R^2 - \pi r^2)$$

$$g(S) = 2r + f(r - 1) + g(S - \pi r^2)$$

где r — радиус максимального вписанного круга в область.

Докажем, что $f(R) \leq 2\pi R^2$ и $g(S) \leq 2S$.

$$f(R) \leq 2r + 2\pi(r - 1)^2 + 2(\pi R^2 - \pi r^2) \leq 2\pi R^2 + (4 - 4\pi)r + 2\pi \leq 2\pi R^2$$

$$g(S) \leq 2r + 2\pi(r - 1)^2 + 2(S - \pi r^2) \leq 2S + (4 - 4\pi)r + 2\pi \leq 2S$$

Задача F. False figures

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Эта задача является валидатором тестов к предыдущей задаче. Самое сложное уже проверили, вам предлагается всего лишь проверить вложенность кругов.

Представим таблицу $\{a_{ij}\}$ из 1 000 строк и 1 000 столбцов. Дано Q запросов вида:

$$i_k \ j_k \ r_k, \ 1 \leq k \leq Q$$

Каждый из них определяет область из элементов таблицы a_{ij} таких, что $(i - i_k)^2 + (j - j_k)^2 \leq r_k^2$.

Для удобства будем называть эту область кругом с центром в ячейке (i_k, j_k) и радиусом r_k .

Пара запросов k и l ($k < l$) считается невалидной, если круги k и l имеют общие ячейки, и существует круг t : $k < t \leq l$, который не содержится в круге k . Круги различных запросов могут совпадать.

Определить, есть ли среди запросов хотя бы одна невалидная пара.

Формат входного файла

В первой строке число Q — количество запросов. Далее в Q строках описаны запросы по три целых числа: i_k, j_k, r_k , в каждой строке — номера строки и столбца центральной ячейки и радиус круга. Ячейки нумеруются с 1.

Формат выходного файла

Если есть хотя бы одна невалидная пара запросов, вывести номера этих запросов в произвольном порядке. Если таких пар много, разрешается вывести любую из них. Если невалидных пар нет, вывести “Ok”.

Ограничения

$$1 \leq Q \leq 10^6$$

$$1 + r_k \leq i_k \leq 1000 - r_k$$

$$1 + r_k \leq j_k \leq 1000 - r_k$$

$$0 \leq r_k < 500$$

Пример

stdin	stdout
6 10 10 5 10 10 4 5 10 0 10 5 0 10 15 0 15 10 0	Ok
2 6 6 5 11 11 5	2 1

Разбор задачи F. False figures

Предположим, что все круги вложены правильно. Обойдем их в порядке вложенности. Если нарушение есть, оно обнаружится при рассмотрении некоторого круга.

Пусть мы находимся в некотором запросе. Будем рассматривать все круги, центр которых лежит внутри текущего. Если для какого-либо из них обнаружилось нарушение, дальнейшее рассмотрение не имеет смысла. Если нарушение не обнаружилось, мы имеем список кругов, которые надо проверить на вложенность в текущий круг и на отсутствие общих клеток между любой парой из них. Разобьем текущий круг на строки и для каждого из его вложенных кругов определим отрезок строк, которые он пересекает. После чего, обрабатывая каждую строку, мы можем посчитать, нет ли там пересечения областей каких-либо двух внутренних кругов. А также, не выходит ли некоторый внутренний круг за границу исходного. Если все это соблюдается, это означает, что все внутренние круги действительно вложены в текущий и они нас больше не волнуют. Если обнаружится нарушение, это будет на этапе проверки на пересечение внутренних кругов.

Согласно предыдущей задаче можно тратить по $O(r_i)$ операций на каждый запрос, чтобы итоговая сложность была $O(S)$, где S — площадь фигуры. Из-за проверки на пересечения внутренних кругов в каждой строке, добавляется еще \log при сортировке.

Задача G. Grouping

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дано N целых чисел $\{a_i\}$. Пусть $\{b_j\}$ — такие K чисел (не обязательно целых), что:

$$S = \sum_{i=1}^N \min_{1 \leq j \leq K} |a_i - b_j|$$

И S — минимально возможное.

Найти S .

Формат входного файла

В первой строке два числа N и K . Во второй строке ровно N целых чисел — $\{a_i\}$.

Формат выходного файла

Единственное вещественное число — S с абсолютной или относительной погрешностью не более 10^{-8} .

Ограничения

$$1 \leq N \leq 5\,000$$

$$1 \leq K \leq N$$

$$0 \leq a_i \leq 400\,000$$

Пример

stdin	stdout
5 3 1 5 7 10 14	5.0

Пояснение

В качестве $\{b_i\}$ имеет смысл взять $\{1, 7, 14\}$.

Разбор задачи G. Grouping

Перефразируем задачу таким образом, что нам нужно разбить все точки на K групп и в каждой из них выбрать b_j . Очевидно, что при оптимальном разбиении все точки одной группы располагаются подряд. Для каждой отдельной группы b_j будет медианой этой группы. Поэтому для каждого интервала $[i, j]$ отсортированного исходного набора можно предсчитать оптимальное b и даже суммарный штраф всей группы.

Тогда всю задачу можно решать динамикой: $d[i][j]$ — минимальный штраф при разбиении первых i точек на j групп. Пересчитывается так:

$$d[i][j] = \min_{1 \leq p \leq i} d[p-1][j-1] + cost(p, i)$$

где $cost(p, i)$ — штраф группы, созданной из всех точек от p до i .

Такая динамика очевидно работает за $O(N^3)$. Пусть $p[i][j]$ — p , при котором достигается оптимальное $d[i][j]$. Тогда соблюдается неравенство:

$$p[i-1][j] \leq p[i][j] \leq p[i][j+1]$$

Теперь, если перебирать $p[i][j]$ в пределах таких границ, суммарное количество операций будет:

$$\begin{aligned} \sum_{i=1}^N \sum_{j=1}^K (p[i][j+1] - p[i-1][j]) &= \sum_{i=1}^N \sum_{j=2}^K p[i][j] + \sum_{i=1}^{N-1} \sum_{j=1}^K p[i][j] = \\ &= \sum_{j=2}^K p[N][j] - \sum_{i=1}^{N-1} p[i][1]. \end{aligned}$$

Поскольку $p[i][j]$ принимает значение от 1 до i , то сумма не превысит NK . Поэтому сложность $O(NK)$.

Задача Н. Hidden triangles

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 4 с
 Ограничение по памяти: 512 Мб

На плоскость положили N треугольников по порядку от 1 до N . Вся внутренняя область каждого треугольника является непрозрачной и закрывает все, что находится под ней.

Определить, какие из треугольников остались видны на плоскости. То есть имеют область положительной площади, не накрытую сверху никаким другим треугольником.

Формат входного файла

В первой строке число N — количество треугольников. Далее в N строках перечислены треугольники в том порядке, в котором они выкладывались на плоскость. Каждый треугольник описывается шестью целыми числами $x_{i1}, y_{i1}, x_{i2}, y_{i2}, x_{i3}, y_{i3}$ — координатами его вершин. Все треугольники невырожденные. Любая сторона одного треугольника имеет не более одной общей точки с любой стороной другого треугольника.

Формат выходного файла

В первой строке вывести количество видимых треугольников. Во второй строке перечислить их номера в произвольном порядке.

Ограничения

$$1 \leq N \leq 500$$

$$-1\,000 \leq x_{ij}, y_{ij} \leq 1\,000, \text{ для } 1 \leq i \leq N, 1 \leq j \leq 3$$

Пример

stdin	stdout
3	2
1 0 4 0 0 3	2 3
-2 1 5 -2 3 4	
-2 2 4 1 2 4	

Разбор задачи Н. Hidden triangles

Построим планарный граф, вершинами которого будут все вершины треугольников и точки пересечения их сторон, а ребрами — отрезки сторон треугольников между вершинами графа. Построить такой граф можно за $O(N^2 \log(N))$, поскольку всего пересечений $O(N^2)$, а для фиксированной стороны треугольника все пересечения на ней можно отсортировать по порядку следования от одной вершины к другой за $O(N \log(N))$. Выделим грани этого планарного графа путем сортировки ребер по углу за $O(N^2 \log(N))$.

Каждая грань принадлежит нескольким треугольникам, а виден среди них треугольник с наибольшим номером. Запустим поиск в глубину по граням, при этом заведем глобальное множество, которое на каждом этапе обхода будет хранить номера всех треугольников, содержащих текущую грань. Для каждой грани возьмем максимум в этом множестве, чтобы определить номер треугольника, который является верхним среди всех, покрывающих эту грань. Заметим, что при переходе к соседней грани по ребру в это множество либо добавляется треугольник, сторона которого содержит это ребро, либо наоборот удаляется, если он там присутствовал. В качестве структуры данных для хранения такого множества можно использовать set.

Поиск в глубину следует запускать от внешней грани планарного графа, которая не принадлежит ни одному треугольнику.

Заметим, что может получиться несколько компонент связности. Тогда для двух компонент возможны два случая:

- Одна из компонент полностью лежит во внутренней грани другой.
- Обе компоненты не имеют общей площади (Обе компоненты лежат во внешних гранях друг друга).

Во втором случае можно рассматривать обе компоненты независимо. В первом же, можно рассматривать независимо, только если компонента, лежащая внутри грани, полностью выше этой грани. Для этого изначально имеет смысл удалить все треугольники, полностью содержащиеся в других.

Треугольник считается видимым, если он был самым верхним при рассмотрении хотя бы одной грани.

Задача I. Interactive

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

<http://codeforces.ru/blog/entry/4037>

Alex_KPR: — ...начиная где-то с декартовых деревьев, FFT и так далее, написание и отладка кода становятся довольно занудными задачами...

Kunyavsky: — А можно я придерусь к мелочи. Что делают в одном ряду по времени написания декартова дерева и FFT?

Alex_KPR: — это первое, что пришло в голову, где нужно много писать и есть где накосячить

Kunyavsky: — Я же сказал: придираюсь к мелочам. Просто на мой взгляд рядом поставлены вещи, одна из которых несравнимо легче по написанию

homo_sapiens: — Полностью согласен Фурье пишется вдвое может даже втрое быстрее чем дерево (но его обычно пихать надо руками и ногами)

Kunyavsky: — Я имел ввиду с точностью наоборот. Все таки очень специфичная тема.

homo_sapiens: — Видимо и правда на вкус и цвет...

Пришло время узнать, что же проще.

Вам предлагаются две последовательности из N целых чисел $\{x_i\}$ и $\{y_i\}$.

Если вам больше нравится FFT, представьте, что это коэффициенты двух многочленов:

$$P(z) = \sum_{i=0}^{N-1} x_i z^i$$

$$Q(z) = \sum_{i=0}^{N-1} y_i z^i$$

Найдите коэффициенты многочлена $P(z) \cdot Q(z)$.

Если же вам больше нравятся Декартовы деревья, представьте, что $\{x_i\}$ — ключи, а $\{y_i\}$ — приоритеты. Постройте Декартово дерево путем вставки в него последовательно всех N элементов (x_i, y_i) . После каждой вставки выводите высоту получившегося дерева. Высота дерева — количество ребер на длиннейшем пути от какой-либо вершины к корню.

Для тех, кто выбрал FFT, напомним, что Декартово дерево — это двоичное дерево поиска по ключам, содержащимся в вершинах (x_i) . И одновременно куча по приоритетам в вершинах (y_i) . То есть, для каждой вершины дерева выполняется свойство, что все вершины из левого поддерева имеют ключи меньше, чем в данной вершине, а все вершины из правого поддерева — большие. А также приоритет данной вершины больше приоритета ее сыновей.

Независимо от того, что же вы выбрали, автор убедительно просит не использовать в этой задаче `prewritten code`.

Формат входного файла

В первой строке число N . Во второй строке N различных целых чисел — $\{x_i\}$. В третьей строке N различных целых чисел — последовательность $\{y_i\}$.

Формат выходного файла

Если вы выбрали FFT, то выведите $2N-1$ целых чисел — коэффициенты результирующего многочлена. Если же вы выбрали Декартово дерево, то выведите N целых чисел — высоты дерева после вставки в него каждой вершины.

Вообще, вы можете вывести и то, и то, но тогда оба результата будут проверяться на корректность.

Ограничения

$$1 \leq N \leq 50\,000$$

$$1 \leq x_i \leq 50\,000$$

$$1 \leq y_i \leq 50\,000$$

гарантируется, что высота дерева не превысит 50

Пример

stdin	stdout
6 4 1 2 3 6 5 1 6 3 4 5 2	4 25 20 34 54 71 72 58 56 37 10
6 4 1 2 3 6 5 1 6 3 4 5 2	0 1 2 2 3 4

Разбор задачи I. Interactive

Задача решается одним из двух стандартных алгоритмов. Оба работающих за $O(N \log(N))$.

Задача J. Journey

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Говорят, планета J-Рах имеет форму прямоугольного параллелепипеда с длиной A , шириной B и высотой C . Человеку, живущему в вершине, захотелось путешествовать. Он выбрал в качестве цели самую удаленную по поверхности точку планеты. Найдите расстояние, которое ему придется преодолеть при условии, что он добирается к цели кратчайшим путем.

Формат входного файла

В единственной строке три целых числа A , B , C .

Формат выходного файла

Единственное вещественное число — расстояние, с абсолютной или относительной погрешностью не более 10^{-8} .

Ограничения

$$1 \leq A, B, C \leq 1\,000$$

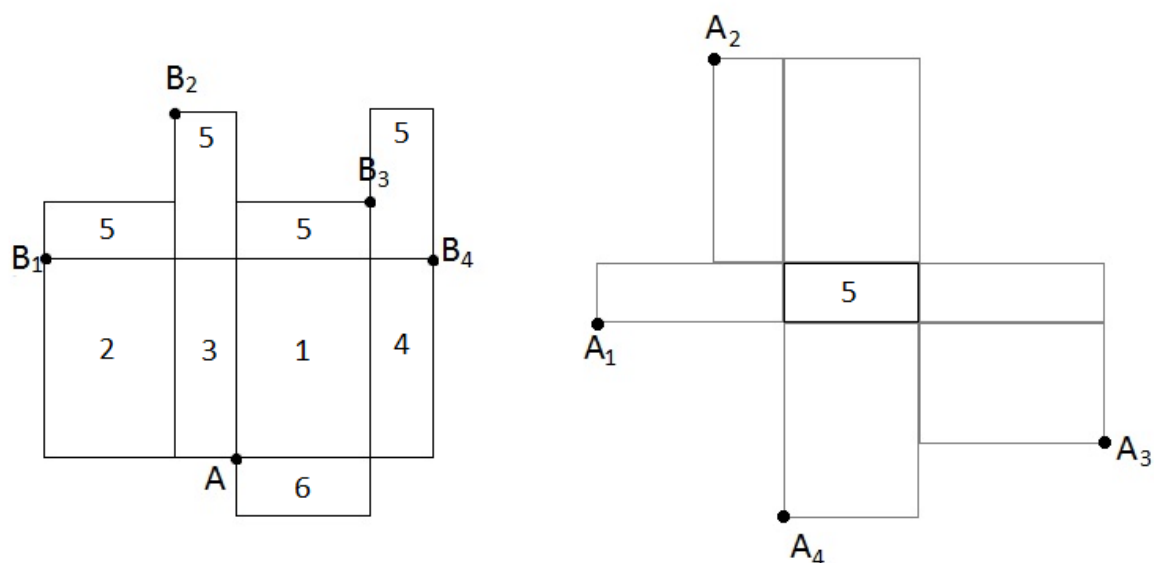
Пример

stdin	stdout
1 1 1	2.2360679774998

Разбор задачи J. Journey

Прежде всего, ответ не лежит в противоположной вершине. Лежит только в случае с кубом. Однако противоположная вершина является одной из самых удаленных от текущей вершины точек. Поэтому используем ее для поиска ответа.

Рассмотрим развертку параллелепипеда (изображение слева).



Точка A — исходная вершина, точки B_1, B_2, B_3, B_4 — те точки развертки, куда попадает противоположная вершина. Очевидно, что B_4 — ближайшая из них. Из этого можно сделать вывод, что грани 1, 2, 3, 4, 6 находятся ближе к вершине A чем вершина B_4 . То есть ответ лежит в грани 5.

Рассмотрим грань 5 отдельно (изображение справа). Теперь вершины A_1, A_2, A_3, A_4 обозначают возможные положения на развертке исходной вершины A относительно грани 5.

Требуется найти на грани 5 самую удаленную точку от этих вершин. Это можно сделать бинарным поиском по ответу, для каждого расстояния d определяя, можно ли полностью покрыть грань 5 кругами с центрами в точках A_1, A_2, A_3, A_4 и радиусом d .

Задача К. Knuth knows

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	512 Мб

Дан ациклический ориентированный граф из N вершин и K ребер. Требуется найти количество ребер в его транзитивном замыкании.

Транзитивное замыкание графа G — граф G' , состоящий из множества вершин исходного графа G и множества ребер (u, v) таких, что существует путь из вершины u в вершину v в графе G .

Кнут знает, как решать эту задачу, а Вы?

Формат входного файла

В первой строке два числа N и K . Далее K строк, в каждой из которых по два целых числа a_i и b_i , означающих наличие ребра, ведущего из вершины a_i в b_i . Граф не содержит петель, циклов и кратных ребер.

Формат выходного файла

Вывести единственное число — количество ребер в транзитивном замыкании.

Ограничения

$$1 \leq N \leq 50\,000$$

$$0 \leq K \leq 50\,000$$

$$1 \leq a_i, b_i \leq N$$

Пример

stdin	stdout
5 6 1 2 2 3 3 5 4 5 1 5 1 3	7

Разбор задачи К. Knuth knows

Ребро (u, v) нужно добавить в граф, если его еще нет, и существует путь из вершины u в вершину v . Обойдем граф обходом в глубину. Для текущей вершины u будем определять $S[u]$ — множество всех вершин, до которых существует путь из вершины u . $S[u] = \bigcup S[v_i] \cup v_i$, где v_i — сын вершины u .

Множества S надо хранить в виде битовой маски. Тогда объединение будет происходить за $N/32$ операций.

Общая сложность $O(NK/32)$.

Задача L. Lake

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Саша находится в начале координат. Но очень хочет попасть на берег озера, который почему-то находится в точке (x, y) . Возможно, потому что там растет дерево. Он купил в ближайшем магазине сапоги-скороходы, с помощью которых может шагнуть на расстояние ровно d .

Определите, какое минимальное количество шагов потребуется, чтобы оказаться в желаемом месте. Точка (x, y) не совпадает с началом координат.

Саша может наступать в любую точку нашего плоского мира.

Формат входного файла

В единственной строке три целых числа x, y, d .

Формат выходного файла

Единственное число — минимальное количество шагов.

Ограничения

$$-10\,000 \leq x, y \leq 10\,000$$

$$1 \leq d \leq 10\,000$$

Пример

<code>stdin</code>	<code>stdout</code>
6 3 2	4

Разбор задачи L. Lake

Пусть расстояние до точки (x, y) равно $T = k \cdot d + l, 0 \leq l < d$. Если $l = 0$, то ответ равен k . Теперь пусть $l > 0$. Понятно, что тогда нам не хватит k шагов. Докажем, что нам будет достаточно $k + 1$ шага. Для начала сделаем $k - 1$ шаг в направлении точки (x, y) . Затем рассмотрим окружность радиуса d вокруг точки (x, y) . Кратчайшее расстояние до точки на этой окружности от нашего текущего положения равно l , что меньше чем d , а длиннейшее равно $2d + l$. Значит, на окружности есть точка, расстояние от которой до

нашей текущей точки равно d . Походим в эту точку, и оттуда в финальную точку. Таким образом, нам хватило $k + 1$ шага.

Отдельно рассмотрим случай, когда расстояние до целевой точки меньше d , аналогичными рассуждениями про окружность можно достичь финальной точки за два хода.

Задача М. Match them up

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

В этой задаче вам придется искать максимальное паросочетание. Да не простое, а лексикографически минимальное.

Дан двудольный граф из N вершин в левой доле, N вершин в правой доле и K ребер между ними.

Максимальное паросочетание — максимальное по мощности подмножество ребер графа M , такое, что любая вершина графа инцидентна не более чем одному ребру из M .

Пусть множество ребер $\{(u_i, v_i)\}$ — максимальное паросочетание, где u_i — вершины из левой доли, v_i — вершины из правой доли. Выпишем все вершины в одну последовательность в порядке: $u_1, v_1, u_2, v_2, \dots, u_m, v_m$.

Найдите максимальное паросочетание с лексикографически минимальной последовательностью вершин.

Формат входного файла

В первой строке два числа N и K — количество вершин в каждой из долей и количество ребер. Далее K строк с описанием ребер. Каждое ребро описывается парой чисел a_i и b_i , где a_i — вершина из левой доли, а b_i — из правой доли.

Формат выходного файла

В первой строке выведите размер максимального паросочетания m . Далее m строк по два числа u_i, v_i в каждой, где u_i — вершина из левой доли, а v_i — вершина из правой доли, соответствующие i -ому ребру паросочетания.

Ограничения

$$1 \leq N \leq 10^3$$

$$0 \leq K \leq 10^5$$

$$1 \leq a_i, b_i \leq N$$

Пример

stdin	stdout
3 5	3
1 2	1 3
1 3	2 1
3 2	3 2
2 3	
2 1	

Разбор задачи М. Match them up

Найдем некоторое максимальное паросочетание. Будем пытаться получить из него лексикографически минимальное. Возьмем первую вершину и попробуем вставить ее в паросочетание, если ее там еще нет. Если удастся вставить, то пробуем в качестве пары подобрать ей минимальную вершину из правой доли. Все это делается с условием, что величина паросочетания не меняется. Если первую вершину вставить удалось, то она и ее пара фиксируются и больше не могут быть изменены. Аналогичные действия производятся с вершиной 2, потом 3 и т.д.

Пусть очередная вершина левой доли a . Проверим, можно ли вставить ее в паросочетание и выбрать ей минимальную пару. Если вершина a сейчас не в паросочетании, то выберем такую вершину b из правой доли, что b минимальная, еще не зафиксированная, и существует ребро (a, b) . Освободим b и ее текущую пару и назначим в паросочетании новую пару (a, b) , и зафиксируем ее. Очевидно, размер паросочетания не изменится, и среди всех вариантов мы выбрали минимальный.

Если вершина a принадлежит паросочетанию, то основная задача — подобрать ей минимальную пару. Для этого освободим ее и ее пару. Теперь нужно восполнить потерю одного ребра в паросочетании. Попробуем сначала увеличить паросочетание, не используя вершину a . Это делается для того, чтобы после этого была возможность для a выбрать любую не зафиксированную вершину, связанную с ней. Если паросочетание восстановилось без помощи a , то так и делаем. По сути это эквивалентно первому рассмотренному случаю. Если паросочетание не увеличилось, то мы будем искать увеличивающийся путь прямо из вершины a . Будем стараться искать его жадно, подбирая в обходе в глубину пару для a в порядке возрастания вершин правой доли. Увеличивающийся путь точно найдется, так

$$A_i = (1\,234\,567 \cdot A_{i-1} + 7\,654\,321) \bmod 1\,000\,000\,007, \text{ при } 2 \leq i \leq Q$$

$$r_i = A_i \bmod N + 1$$

$$c_i = A_i \bmod (2r_i - 1) + 1$$

$$k_i = A_i \bmod (N - r_i + 1) + 1$$

Формат входного файла

В первой строке N и Q — количество строк исходного треугольника и количество запросов. Далее в N строках идет описание таблицы. В i -той из них ровно $2i - 1$ цифр — элементы таблицы (между цифрами нет пробелов).

Формат выходного файла

Поскольку запросов много, выведите одно число — сумму всех ответов на запросы.

Ограничения

$$1 \leq N \leq 10^3$$

$$0 \leq Q \leq 5 \cdot 10^6$$

Пример

stdin	stdout
3 5 1 234 56789	42

Пояснение

1
234

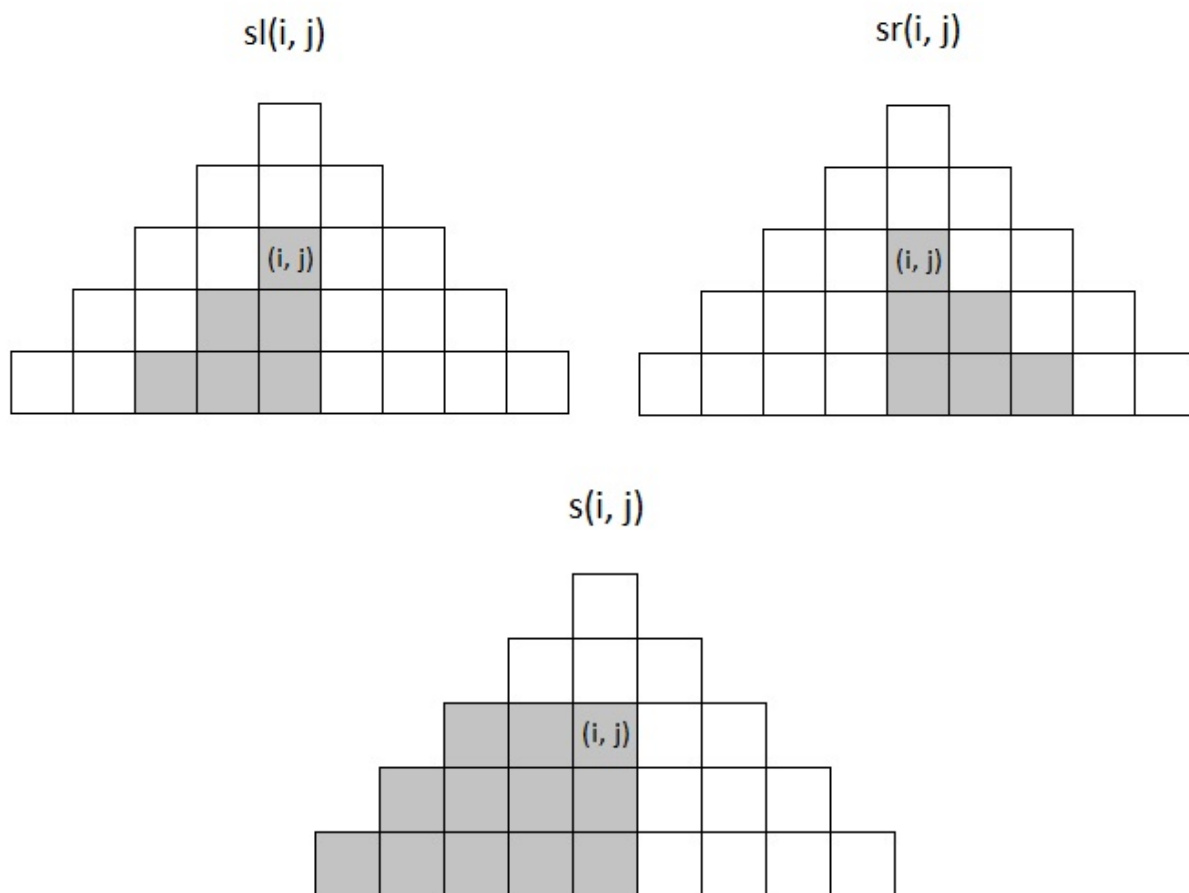
Исходный треугольник: 56789

3

Первый запрос: 678, сумма 24.
 Второй запрос: 8, сумма 8.
 Третий запрос: 6, сумма 6.
 Четвертый запрос: 1, сумма 1.
 Пятый запрос: 3, сумма 3.

Разбор задачи N. Need for sum thing

Для каждой клетки (i, j) посчитаем три величины: $sl(i, j)$, $sr(i, j)$ и $s(i, j)$.



$sl(i, j)$ — сумма элементов левой лестницы, $sr(i, j)$ — сумма элементов правой лестницы, $s(i, j)$ — сумма всех элементов слева и снизу.

Тогда, чтобы ответить на запрос i, j, k , необходимо вычислить:

$$ans = sl(i, j) - sl(i + k, j) + sr(i + 1, j + 2) - sr(i + k, j + 2k) + s(i + k, j + 2k - 1) - s(i + k, j).$$

Задача O. Open air

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Есть полный набор доминошек, на каждой из которых записано по два числа от 0 до N (N — четное). Причем каждая пара чисел $\{a, b\}$ встречается ровно один раз. Нетрудно посчитать, что количество доминошек в наборе — $\frac{(N + 1)(N + 2)}{2}$.

Требуется расположить все доминошки горизонтально в $N + 1$ ряд так, чтобы суммы чисел, записанных на всех доминошках в каждом ряду, были равны. Гарантируется, что это всегда можно сделать.

Формат входного файла

В единственной строке четное число N .

Формат выходного файла

Вывести $N + 1$ строку по $N + 2$ числа в каждой: $a_1, b_1, a_2, b_2, \dots, a_k, b_k$, $k = \frac{N + 2}{2}$, где каждая пара a_i, b_i соответствует очередной доминошке. Каждую доминошку нужно использовать ровно один раз. Пару чисел на каждой доминошке можно выводить в любом порядке. Если решений несколько, можете вывести любое из них.

Ограничения

$2 \leq N \leq 100$, N — четное

Пример

stdin	stdout
2	1 2 0 1 1 1 0 2 0 0 2 2

Разбор задачи О. Open air

В каждой строке должна быть фиксированная сумма $S = \frac{N(N + 2)}{2}$. Используем эвристику. Заполним строки случайно и посчитаем суммы в строках. Будем выполнять итерации следующего вида:

Возьмем две случайные доминошки i и j из строк p_i и p_j соответственно. v_i — значение доминошки i , v_j — значение доминошки j . Пусть сумма всех доминошек p_i -ой строки была s_i , а сумма p_j -ой строки — s_j . Поменяем доминошки местами. Тогда сумма p_i -ой строки станет $new_s_i = s_i - v_i + v_j$, а сумма p_j -ой — $new_s_j = s_j - v_j + v_i$.

Если $|new_s_i - S| + |new_s_j - S| < |s_i - S| + |s_j - S|$, то оставляем все как есть, иначе обратно меняем местами доминошки i и j .

В результате большого количества таких итераций мы окажемся в локальном оптимуме. Если он не будет искомым ответом, перемешаем все доминошки заново.

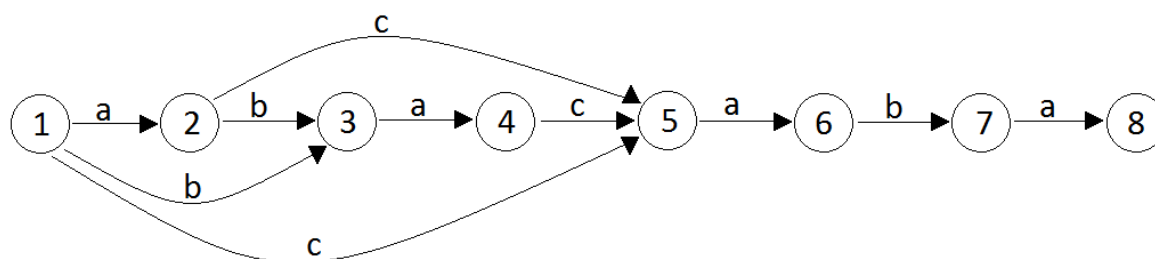
Поскольку суммы на доминошках небольшие и близкие по значению, вариантов требуемого распределения доминошек по строкам очень много. Поэтому подобный способ нахождения ответа работает быстро.

Задача P. Pseudo automaton

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Дана строка S . Построить детерминированный конечный автомат, принимающий все суффиксы строки S (и возможно другие конечные строки). Автомат должен состоять из минимального числа состояний. Каждое состояние автомата объявляется финальным. Начальное состояние имеет номер 1.

На изображении ниже показан автомат из тестового примера для строки “abacaba”



Формат входного файла

В единственной строке слово S , состоящее из строчных латинских букв.

Формат выходного файла

В первой строке два числа N и K — количество состояний и количество переходов. Далее K строк, в каждой из которых по два числа a_i , b_i и буква c_i , означающие наличие перехода из состояния a_i в b_i по букве c_i . Переходы можно выводить в любом порядке. Если решений несколько, можете вывести любое из них.

Ограничения

$$1 \leq |S| \leq 5\,000$$

$$1 \leq a_i, b_i \leq N$$

$$'a' \leq c_i \leq 'z'$$

Пример

stdin	stdout
abacaba	8 10 1 2 a 1 3 b 1 5 c 2 3 b 2 5 c 3 4 a 4 5 c 5 6 a 6 7 b 7 8 a

Разбор задачи P. Pseudo automaton

Пусть N — длина строки S . Построим автомат, который принимает все строки длины от 1 до N . Для этого введем $N + 1$ состояние и $26N$ переходов: из состояния i в состояние $i + 1$ по каждой из букв от 'a' до 'z', при $1 \leq i \leq N$. Очевидно, такой автомат в том числе будет принимать и все суффиксы строки S .

Задача Q. Quiz

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Каждый из вас скорее всего знаком с детской игрой “пятнашки”. В этой задаче требуется найти решение для некоторой позиции.

Игра заключается в следующем: есть квадратное поле $N \times N$, разбитое на клетки 1×1 . В во всех клетках кроме одной есть фишки, на каждой из которых записано число от 1 до $N^2 - 1$. Каждое число встречается ровно один раз. На этом поле можно осуществлять ходы фишками, а именно, за

один ход можно передвинуть одну из соседних с пустым местом фишек на это пустое место. При этом на месте фишки образуется пустое место. Фишки не могут покидать пределы поля.

Решить головоломку — значит расположить фишки в определенном порядке:

$$\begin{array}{cccccc}
 1 & 2 & \dots & N-1 & N \\
 N+1 & N+2 & \dots & 2N-1 & 2N \\
 \vdots & & & & \vdots \\
 N^2 - N + 1 & N^2 - N + 2 & \dots & N^2 - 1 &
 \end{array}$$

Пустое место должно оказаться в последней клетке последней строки. Для обычных пятнашек это выглядит как:

$$\begin{array}{cccc}
 1 & 2 & 3 & 4 \\
 5 & 6 & 7 & 8 \\
 9 & 10 & 11 & 12 \\
 13 & 14 & 15 &
 \end{array}$$

Дана позиция, найти последовательность ходов (не обязательно кратчайшую, возможно пустую), которая ее решает. Либо сказать, что позиция не имеет решения.

Формат входного файла

В первой строке число N — размер поля. Далее N строк по N чисел в каждой. Числа от 1 до $N^2 - 1$ соответствуют фишкам, 0 соответствует пустому месту. Каждое число от 0 до $N^2 - 1$ встречается ровно один раз.

Формат выходного файла

Если позиция не имеет решения, вывести “No”.

В противном случае в первой строке вывести “Yes”, а во второй — строку из ходов (без пробелов):

‘L’ означает, что на пустое место надо передвинуть фишку, находящуюся слева от него.

‘R’ означает, что на пустое место надо передвинуть фишку, находящуюся справа от него.

‘U’ означает, что на пустое место надо передвинуть фишку, находящуюся сверху от него.

‘D’ означает, что на пустое место надо передвинуть фишку, находящуюся снизу от него.

Количество ходов не должно превышать 2 500 000. Если решений несколько, можете вывести любое из них.

Ограничения

$$2 \leq N \leq 50$$

$$0 \leq a_{ij} \leq N^2 - 1$$

Пример

stdin	stdout
2 0 3 2 1	Yes DRULDR
2 2 1 3 0	No

Разбор задачи Q. Quiz

Решение конструктивное.

Первый этап — собрать каждую из первых $N - 2$ строк. Пусть мы уже полностью собрали $i - 1$ строку и в i -ой строке собрали $j - 1$ элемент. Необходимо поставить элемент $t = (i - 1)N + j$ на свое место (i, j) . Если $j < N$, то перемещаем элемент t на столбец j , после чего поднимаем его на строку i , обходя пустым местом справа от элемента. Если $j = N$ то сдвинем всю строку i на один элемент влево, а первый элемент строки i на клетку вниз. Потом вставим элемент t на позицию $j - 1$ строки i , так если бы мы вставляли элемент $t - 1$. После чего сдвинем строку i на одну клетку вправо, и все ее элементы станут на свои места.

Когда первые $N - 2$ строки собраны, последние две собираются по столбцам. Чтобы собрать столбец j , надо сначала элемент $(N - 1)N + j$ поставить на позицию $(N - 1, j)$, а элемент $(N - 2)N + j$ — на позицию $(N - 1, j + 1)$. Дальше эту конструкцию сдвинуть вниз, чтобы элементы стали на свои места. Перейти к следующему столбцу.

Когда останутся четыре элемента, надо их циклически сдвигать, пока элемент $N^2 - 1$ не окажется на месте $(N, N - 1)$, а пустое место — на (N, N) . Тогда либо пятнашки будут собраны, либо будет понятно, что их собрать невозможно.

На каждый элемент нужно $O(N)$ операций, поэтому всего необходимо порядка $O(N^3)$ команд.

Задача R. Reduction

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Дана строка S и список M , состоящий из N слов, каждое из которых длины L . За одну операцию можно выбрать некоторую подстроку строки S , если такая встречается в качестве слова в списке M , и вырезать из строки S . После чего оставшиеся части строки S , если такие есть, склеиваются. Определить, за какое минимальное количество операций можно уничтожить всю строку S . Гарантируется, что это можно сделать.

Формат входного файла

В первой строке слово S . Во второй строке целое число N — количество слов в списке. Далее N строк, в каждой из которых по одному слову из списка M . Все слова состоят только из строчных латинских букв.

Формат выходного файла

Одно число — минимальное количество операций, необходимое для уничтожения строки S .

Ограничения

$1 \leq |S| \leq 100$; $1 \leq N \leq 100$; $1 \leq |M_i| \leq 100$; $1 \leq L \leq |S|$

Пример

<code>stdin</code>	<code>stdout</code>
abacabada	3
4	
aba	
aca	
ada	
abb	

Разбор задачи R. Reduction

Нужно заметить, что все слова из списка одинаковой длины, и что гарантируется существование решения. Следовательно, ответ равен $\frac{|S|}{L}$.

День второй (17.02.2013 г.) Контеcт Бондаренко Натальи Павловны

Об авторе...

Бондаренко Наталья Павловна, кандидат физико-математических наук, доцент Саратовского государственного университета. На протяжении 2005–2010 гг. капитан команды Saratov SU #1, представлявшей СГУ в ACM ICPC. В подготовке контеcта мне помогали мои товарищи по команде — Дмитрий Матов и Станислав Пак.



Основные достижения:

- I диплом заключительного этапа Всероссийской олимпиады школьников по информатике в 2005 г.;
- абсолютное 1 место в полуфинале ACM ICPC, чемпион России по программированию (в составе команды СГУ) в 2008 г.; золотая медаль ACM ICPC 2009, серебряная медаль ACM ICPC 2010;
- финалистка Google Code Jam 2008 и 2011, VK Cup 2012, Challenge 24 2012;
- 2 место в Russian Code Cup 2012;
- участница всех 12 сезонов Открытого Кубка имени Е.В. Панкратьева по программированию, в числе победителей во всех сезонах, начиная с III-го.

Теоретический материал. Потоки в сетях

Задача о максимальном потоке

Сетью называется ориентированный граф $G = (V, E)$, каждому ребру $(u, v) \in E$ которого поставлено в соответствие число $c(u, v) \geq 0$, называемое *пропускной способностью* ребра. В случае $(u, v) \notin E$ мы полагаем $c(u, v) = 0$. В графе выделены две вершины: *исток* s и *сток* t . Потоком в сети G назовем функцию $f : V \times V \rightarrow R$, удовлетворяющую трем условиям:

1. *Ограничение, связанное с пропускной способностью:* $f(u, v) \leq c(u, v)$, для всех u, v из V .
2. *Антисимметричность:* $f(u, v) = -f(v, u)$ для всех u, v из V .
3. *Сохранение потока:* $\sum_{v \in V} f(u, v) = 0$ для всех u из V , кроме s и t .

Удобно представлять себе G как сеть труб, по которым некоторое вещество движется от истока (где оно производится с некоторой постоянной скоростью) к стоку (где оно потребляется с той же скоростью), не накапливаясь в промежуточных вершинах.

Величина потока определяется как сумма

$$|f| = \sum_{v \in V} f(s, v).$$

Задача состоит в нахождении в заданной сети G потока максимальной величины.

Остаточные сети

Для любой пары вершин u и v сети G , в которой задан поток f , определим *остаточную пропускную способность*: $c_f(u, v) = c(u, v) - f(u, v)$. Сеть $G_f = (V, E_f)$, $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$, называется *остаточной сетью* сети G , порожденной потоком f . Пропускные способности в G_f равны c_f .

Утверждение 1. Пусть f — поток в сети G и f' — поток в остаточной сети G_f . Тогда $f + f'$ — поток в сети G величины $|f + f'| = |f| + |f'|$.

Утверждение доказывается путем проверки свойств 1–3 для потока $f + f'$.

Назовем *дополняющим путем* простой путь w из s в t в сети G_f и *остаточной пропускной способностью* величину

$$c_f(w) = \min\{c_f(u, v) : (u, v) \in w\}.$$

По дополняющему пути p можно пустить дополнительный поток

$$f_w(u, v) = \begin{cases} c_f(p), & (u, v) \in w, \\ -c_f(p), & (v, u) \in w, \\ 0, & \text{в остальных случаях.} \end{cases}$$

величины $|f_w| = c_f(w) > 0$, увеличив тем самым поток в сети G : $f := f + f_w$.

Утверждение 2. Поток f является максимальным тогда и только тогда, когда в остаточной сети не существует дополняющих путей.

Необходимость утверждения очевидна из предшествующих построений: если в остаточной сети есть дополняющий путь w , мы можем увеличить поток f , а значит, он не максимален. Для доказательства достаточности рассмотрим максимальный поток F и некоторый поток f меньшей величины: $|f| < |F|$. Нетрудно показать, что $f' = F - f$ является потоком в остаточной сети G_f и $|f'| > 0$. Значит, в остаточной сети любого не максимального потока существует дополняющий путь.

Метод Форда-Фалкерсона

```

begin
     $f := 0$ 
    while существует дополняющий путь  $w$  в остаточной сети  $G_f$ 
    do
         $f := f + f_w$ 
    end
end

```

Пусть $n = |V|$, $m = |E|$. Время работы метода в случае целых пропускных способностей оценивается как $O(|f|m)$. Алгоритм Эдмондса-Карпа представляет собой реализацию метода Форда-Фалкерсона, при которой дополняющий путь ищется обходом в ширину, он работает за $O(nm^2)$. Это верхняя оценка. На практике алгоритм эффективно работает на 100–200 вершинах. Существуют более эффективные алгоритмы построения максимального потока на $O(n^3)$.

Максимальный поток минимальной стоимости

Рассмотрим сеть G с истоком s и стоком t , в которой каждое ребро $(u, v) \in E$ имеет пропускную способность $c(u, v)$ и цену $p(u, v)$. Стоимостью потока f называется величина

$$p(f) = \sum_{\substack{u, v \in V \\ f(u, v) > 0}} p(u, v) f(u, v).$$

Задача заключается в нахождении потока f минимальной стоимости среди потоков, максимальных по величине.

Для каждого ребра (u, v) сети G добавим обратное ребро (v, u) с пропускной способностью $c(v, u) = 0$ и ценой $p(v, u) = -p(u, v)$. Заметим, что обратные ребра не будут влиять на стоимость потока (поскольку поток по ним неположителен), однако они будут нам нужны в остаточной сети. Цены ребер при переходе к остаточной сети сохраняются $p_f(u, v) = p(u, v)$.

Построение обратных ребер часто приводит к мультиграфу. Например, если между двумя вершинами изначально существуют ребра (u, v) и (v, u) , то после построения к ним обратных ребер между вершинами u и v будет четыре ребра. При реализации в таких случаях удобнее хранить ребра не матрицей смежности, а списками.

Утверждение 3. Пусть f — поток в сети G , f' — поток в остаточной сети G_f . Тогда $p(f + f') = p(f) + p(f')$.

Утверждение 4. Поток f является *оптимальным* (т.е. имеет наименьшую стоимость среди потоков такой же величины) тогда и только тогда, когда остаточная сеть G_f не содержит циклов отрицательного веса (стоимости).

Действительно, если остаточная сеть содержит цикл отрицательного веса w , пусть по нему поток f_w .

Тогда $|f_w| = 0$ и $p(f_w) = c(f_w) \times (\text{вес цикла}) < 0$. Согласно утверждениям 1 и 3, $|f + f_w| = |f|$ и $p(f + f_w) < p(f)$, т.е. найден поток меньшей стоимости и f не является оптимальным.

Для доказательства достаточности рассмотрим разность оптимального потока F и некоторого другого потока f такой же величины, но большей стоимости. Нетрудно показать, что их разность является потоком нулевой величины (*циркуляцией*) в остаточной сети G_f и содержит цикл отрицательного веса.

begin

 Строим произвольный максимальный поток (алгоритмом
 Эдмондса-Карпа)

while *существует цикл отрицательного веса w в остаточной
сети G_f* **do**

$f := f + f_w$

end

end

Цикл отрицательного веса можно искать алгоритмом *Форда-Беллмана* за время $O(nm)$. Общая асимптотика работы алгоритма — $O(nm^2CP)$, $C = \max c(u, v)$, $P = \max p(u, v)$. Если каждый раз искать цикл с наименьшей средней стоимостью, то справедлива полиномиальная оценка $O(nm^2 \log n)$.

Метод дополнения вдоль путей минимальной стоимости

Теорема Форда-Фалкерсона. Пусть f — некоторый оптимальный поток в сети G и w — путь минимальной стоимости из s в t в остаточной сети. Тогда для любого δ , $0 \leq \delta \leq c_f(w)$ поток $f + \delta f_w$ оптимален.


```

begin
   $f := 0$ 
  while существует дополняющий путь в остаточной сети  $G_f$  do
     $w :=$  кратчайший в смысле стоимости путь из  $s$  в  $t$ 
     $f := f + f_w$ 
  end
end

```

Каждая итерация выполняется за время работы поиска кратчайшего пути, обозначим его $d(n, m)$. В сетях с целочисленными пропускными способностями справедлива оценка времени работы $O(d(n, m)|f|)$. Для алгоритма Форда-Беллмана $d(n, m) = O(nm)$, для алгоритма Дейкстры $O(n^2)$, $O(m \log n)$ или $O(n \log n + m)$, в зависимости от реализации.

Как мы видим, алгоритм Дейкстры эффективнее, однако он применим только в случае, если все цены ребер положительны. Но даже если все $p(u, v) \geq 0$ в исходной сети, в остаточной сети могут появиться отрицательные цены ребер. Применение алгоритма Дейкстры к графу с отрицательными ребрами (но без циклов отрицательного веса) дает возможность использование *потенциалов Джонсона*.

Припишем каждой вершине v из V некоторое число $\varphi(v)$, называемое *потенциалом*. Будем называть *остаточной стоимостью* ребра (u, v) величину $p(u, v) + \varphi(u) - \varphi(v)$. Заметим, что сумма остаточных стоимостей вдоль любого пути из u в v отличается от суммы стоимостей вдоль того же пути на величину $\varphi(u) - \varphi(v)$, одинаковую для всех путей. Поэтому кратчайший путь между парой вершин в графе с остаточными стоимостями также является кратчайшим в графе с исходными стоимостями, и наоборот.

Выберем потенциалы таким образом, чтобы все остаточные стоимости в остаточной сети были неотрицательны. Возьмем в качестве $\varphi(v)$ минимальное по цене расстояние от истока s до вершины v или $+\infty$, если вершина v недостижима из истока. Заметим, что $\varphi(v) > -\infty$, поскольку нет циклов отрицательного веса. Тогда неотрицательность остаточной стоимости следует из неравенства

$$\varphi(u) + p(u, v) \leq \varphi(v),$$

всегда выполняющегося для кратчайших путей. Вначале потенциалы Джонсона могут быть найдены одним запуском алгоритма Форда-Беллмана или присвоены нулю, если в исходной сети нет отрицательных ребер. При переходе от одной итерации цикла **while** к другой они должны пересчитываться с учетом вновь найденных кратчайших путей из истока $d(v)$ следующим образом: $\varphi(v) := \varphi(v) + d(v)$.

Список литературы

1. Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн. Алгоритмы: Построение и анализ (2005).
2. Ravindra Ahuja, Thomas Magnanti, James Orlin. Network flows (1993).

Задачи и разборы

Задача А. Самая простая задача

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 2 с
 Ограничение по памяти: 64 Мб

Дана сеть из n вершин, пронумерованных числами от 1 до n . Вершина 1 является истоком, вершина n — стоком. Из вершины i существует ориентированное ребро в вершину j для каждой пары вершин $i < j$, причем это ребро имеет пропускную способность $j - i$. Найдите величину максимального потока в этой сети.

Формат входного файла

Входные данные содержат единственное целое число n ($2 \leq n \leq 100$) — количество вершин графа.

Формат выходного файла

Выведите величину максимального потока.

Примеры

<code>stdin</code>	<code>stdout</code>
5	8

Разбор задачи А. Самая простая задача

Задачу можно решить, не применяя потоковые алгоритмы, а используя специальный вид сети. Для каждой вершины i , пустим по ребрам $1 \rightarrow i$ и $i \rightarrow n$ поток наибольшей величины $\min(i - 1, n - i)$. Также пустим поток по прямому ребру из 1 в n . Получим поток величины

$$n - 1 + \sum_{i=2}^{n-1} \min(i - 1, n - i).$$

Это и есть ответ на задачу.

Строго доказать его оптимальность можно, например, рассмотрев остаточную сеть. Заметим, что в ней вершины с номерами $i \leq n/2$ недостижимы из истока, а из вершин $i > n/2$ недостижим сток. При этом вершины $i \leq n/2$ недостижимы из вершин $i > n/2$. Поэтому в остаточной сети не существует пути из 1 в n и найденный поток — максимальный.

Задача В. Оптимальный поток величины 1

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 2 с
 Ограничение по памяти: 64 Мб

Дан ориентированный граф из n вершин и m ребер. Вершина 1 является истоком, вершина n — стоком. Каждому ребру приписана некоторая цена. Найдите в данной сети оптимальный поток величины 1, т.е. поток величины 1, имеющий наименьшую стоимость. Цены ребер могут быть отрицательными, однако известно, что граф не содержит циклов отрицательного веса. Пропускные способности вам не заданы, про них известно, что они не меньше 1.

Формат входного файла

В первой строке заданы числа n и m ($2 \leq n \leq 1000$, $0 \leq m \leq 10000$). Далее идет m строк с тройками целых чисел x, y, p — ребро из x в y имеет цену p ($1 \leq x, y \leq n$, $-1000 \leq p \leq 1000$). В графе могут быть петли и кратные ребра.

Формат выходного файла

Выведите одно целое число — стоимость оптимального потока величины 1. В случае если в сети не существует потока величины 1, выведите *NO*.

Пример

stdin	stdout
6 7 1 2 -3 1 1 8 2 3 1 1 3 1 3 4 -2 3 1 2 4 6 10	6

Разбор задачи В. Оптимальный поток величины 1

Задача представляет собой ни что иное, как задачу о кратчайшем пути во взвешенном графе, переформулированную в терминах потоков. Для ее решения можно использовать алгоритм Форда-Беллмана (время работы $O(nm)$).

Задача С. Максимальный поток в неориентированном графе

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Найдите величину максимального потока в заданном неориентированном графе.

Формат входного файла

Во входном файле заданы один или несколько тестов. Каждый тест начинается строкой, в которой содержится число n ($2 \leq n \leq 100$) — количество вершин графа. В следующей строке записаны три числа s , t и c — номер истока, номер стока, и количество ребер, соответственно. Далее следуют c строк, в каждой из которых содержатся три числа: номера соединенных вершин ребром и его пропускная способность — целое неотрицательное число, не превосходящее 1 000. Между двумя парами вершин может быть более одного ребра, однако петли не допускаются.

Входной файл завершается фиктивным тестом, состоящим из одного числа 0, который обрабатывать не нужно.

Формат выходного файла

Для каждого из тестов выведите величину максимального потока.

Пример

stdin	stdout
4	25
1 4 5	
1 2 20	
1 3 10	
2 3 5	
2 4 10	
3 4 20	
0	

Разбор задачи С. Максимальный поток в неориентированном графе

Решается при помощи алгоритма Эдмондса-Карпа. Нужно учесть два обстоятельства:

1. Граф неориентированный. Каждое неориентированное ребро можно обрабатывать как два ориентированных.
2. Могут быть мультиребра и их общее количество может быть достаточно большим. Нужно суммировать их пропускные способности, а не хранить отдельно в виде списков, чтобы уложиться в ограничения по времени.

Задача D. Задача про минералку

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 с
 Ограничение по памяти: 64 Мб

Всем известно, что финал чемпионата мира по программированию АСМ 2004 проходил в Праге. Помимо своей известной архитектуры и культуры, Прага всемирно известна своей минералкой. Несмотря на то, что чрезмерное употребление минералки вредит здоровью участников, многие команды воспользовались возможностью попробовать отличную минералку по самым низким ценам.

Новая компания-производитель минералки *Пейте везде* планирует распространять свою продукцию в некоторых из n европейских городов. Собственно артезианская скважина расположена около Праги, которой, конеч-

но, мы присвоим номер 1. Для доставки минералки в другие города компания планирует воспользоваться услугами логистической компании *Вози везде*, которая предлагает m маршрутов для перевозки грузов. Каждый маршрут задается номерами городов, которые он соединяет (грузы можно возить в обоих направлениях), его пропускной способностью — количеством литров минералки, которые могут быть перевезены по этому маршруту за один день, и стоимостью перевозки минералки по нему. Вполне может оказаться так, что для доставки минералки в некоторые города может потребоваться (или быть выгодно) использовать несколько маршрутов один за другим или даже доставлять минералку по нескольким разным путям.

В свою очередь, каждый город характеризуется ценой, по которой местные пабы и рестораны готовы платить за литр минералки. Вы можете считать, что спрос на минералку достаточно неограничен в каждом городе, то есть продукт всегда найдет своего покупателя.

Пейте везде пока не планирует продавать минералку непосредственно в Праге, так как там очень высока конкуренция, поэтому пока она планирует только продавать минералку в некоторых других городах. Помогите им определить, какую максимальную прибыль они могут получить за один день.

Формат входного файла

В первой строке входного находятся числа n и m — число городов в Европе, которые рассматривает компания и количество маршрутов доставки соответственно. ($2 \leq n \leq 100$, $1 \leq m \leq 2000$). В следующей строке находятся $n - 1$ число — цены литра минералки в городах $2, 3, \dots, n$ соответственно (целые числа не превосходящий 1000).

Следующие m строк по четыре числа каждая описывают маршруты. Каждый маршрут задается номерами городов, которые он соединяет, его пропускной способностью и цены перевозки одного литра минералки вдоль него (пропускная способность и цена — положительные числа, не превосходящие 1000).

Формат выходного файла

Выведите одно число — максимальный доход, который может получить компания.

Пример

stdin	stdout
4 4	3000
80 50 130	
1 2 80 50	
2 4 40 90	
3 1 40 60	
3 4 30 50	

Компания должна доставлять 80 литров минералки во второй город (доставка по первому маршруту будет стоить 50 за литр, доход 30 за литр, 2400 всего), и 30 литров в четвертый город (лучший путь идет по маршрутам 3 и 4, его стоимость 110 за литр, доход 20 за литр, 600 всего). Доставлять больше минералки в третий город невыгодно, поэтому компания не должна делать этого.

Разбор задачи D. Задача про минералку

Построим сеть естественным образом. В качестве истока выберем вершину 1, в качестве стока добавим дополнительную вершину. Проведем из всех вершин (кроме истока) в сток ребра бесконечной пропускной способности и цены, равной цене продажи минералки, взятой со знаком “минус”. Требуется найти поток минимальной стоимости. Сеть содержит отрицательные ребра и не содержит циклов отрицательного веса. Согласно теореме Форда-Фалкерсона, метод дополнения вдоль путей минимальной стоимости можно использовать для нахождения оптимального потока любой величины. Также нетрудно видеть, что стоимости путей, найденных этим методом, неубывают. Поэтому будем увеличивать поток вдоль минимальных путей, пока он убывает. Учитывая ограничения задачи, запускаем один раз алгоритм Форда-Беллмана, затем — алгоритм Дейкстры с потенциалами.

Задача E. В поисках невест

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 с
 Ограничение по памяти: 64 Мб

Однажды король Флатландии решил отправить k своих сыновей на поиски невест. Всем известно, что во Флатландии n городов, некоторые из

которых соединены дорогами. Король живет в столице, которая имеет номер 1, а город с номером n знаменит своими невестами.

Итак, король повелел, чтобы каждый из его сыновей добрался по дорогам из города 1 в город n . Поскольку, несмотря на обилие невест в городе n , красивых среди них не так много, сыновья опасаются друг друга. Поэтому они хотят добраться до цели таким образом, чтобы никакие два сына не проходили по одной и той же дороге (даже в разное время). Так как король любит своих сыновей, он хочет, чтобы среднее время сына в пути до города назначения было минимально.

Формат входного файла

В первой строке входного файла находятся числа n , m и k — количество городов и дорог во Флатландии и сыновей короля, соответственно ($2 \leq n \leq 200$, $1 \leq m \leq 2000$, $1 \leq k \leq 100$). Следующие m строк содержат по три целых положительных числа каждая — города, которые соединяет соответствующая дорога и время, которое требуется для ее прохождения (время не превышает 10^6). По дороге можно перемещаться в любом из двух направлений, два города могут быть соединены несколькими дорогами.

Формат выходного файла

Если выполнить повеление короля невозможно, выведите на первой строке число -1 . В противном случае выведите на первой строке минимальное возможное среднее время (с точностью 5 знаков после десятичной точки), которое требуется сыновьям, чтобы добраться до города назначения, не менее чем с пятью знаками после десятичной точки. В следующих k строках выведите пути сыновей, сначала число дорог в пути и затем номера дорог в пути в том порядке, в котором их следует проходить. Дороги нумеруются, начиная с единицы, в том порядке, в котором они заданы во входном файле.

Пример

stdin	stdout
5 8 2	3.00000
1 2 1	3 1 5 6
1 3 1	3 2 7 8
1 4 3	
2 5 5	
2 3 1	
3 5 1	
3 4 1	
5 4 1	

Разбор задачи Е. В поисках невест

Фактически нам задан граф и нужно найти k путей из вершины 1 в вершину n , не пересекающихся по ребрам и наименьшей средней длины. Ясно, что вместо средней длины можно минимизировать суммарную длину. Припишем ребрам единичные пропускные способности и стоимости. В силу теоремы Форда-Фалкерсона, метод дополнения вдоль пути минимальной стоимости позволяет нам найти оптимальный поток любой величины. Найдем оптимальный поток величины k , запустив поиск кратчайшего пути в остаточной сети k раз. Можно на каждой итерации использовать алгоритм Форда-Беллмана, это даст асимптотику времени работы $O(nmk)$.

Задача F. Задача о назначениях

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Дана квадратная матрица из натуральных чисел порядка n . Выберите n элементов из этой матрицы, так, чтобы:

1. в каждой строке и в каждом столбце находился ровно 1 выбранный элемент;
2. из всех наборов, удовлетворяющих первому условию, сумма выбранных чисел должна быть наименьшей.

Если таких наборов несколько — выведите любой.

Формат входного файла

В первой строке записано натуральное число n ($1 \leq n \leq 200$). В каждой из следующих n строк записано по n чисел. Каждый элемент матрицы не превосходит 1000.

Формат выходного файла

В первую строку выведите сумму найденных чисел. Во вторую строку выведите n чисел. i -ое число в последовательности должно обозначать такой номер столбца, что на пересечении i -ой строки и данного столбца стоит выбранный элемент. Очевидно, эта последовательность должна быть перестановкой чисел от 1 до n .

Пример

stdin	stdout
5	23
7 10 5 3 7	4 3 1 2 5
10 6 6 10 9	
7 7 7 5 9	
5 1 4 7 7	
5 10 6 5 6	

Разбор задачи F. Задача о назначениях

Это классическая задача на применение максимального потока минимальной стоимости. Построим двудольный граф, вершинами первой доли которого будут строки матрицы, второй доли — столбцы. Проведем ориентированные ребра пропускной способности 1 из первой доли во вторую и цены, равной числу на пересечении соответствующих строки и столбца матрицы. Добавим дополнительные вершины — исток и сток. Проведем из истока ребра во все вершины первой доли пропускной способности 1 и цены 0 и аналогичные ребра из всех вершин второй доли в сток. Любой максимальный поток в этой сети имеет величину n и соответствует некоторому наибольшему паросочетанию в двудольном графе строк и столбцов. Решение исходной задачи дает максимальный поток минимальной стоимости. При помощи алгоритма Дейкстры с потенциалами он может быть найден за $O(n^3)$. Можно применять и менее эффективные алгоритмы.

Задача G. План эвакуации

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 с
 Ограничение по памяти: 64 Мб

В городе есть муниципальные здания и бомбоубежища, которые были специально построены для эвакуации служащих в случае ядерной войны. Каждое бомбоубежище имеет ограниченную вместимость по количеству людей, которые могут в нем находиться. В идеале все работники из одного муниципального здания должны были бы бежать к ближайшему бомбоубежищу. Однако, в таком случае, некоторые бомбоубежища могли бы переполниться, в то время как остальные остались бы наполовину пустыми.

Чтобы разрешить эту проблему Городской Совет разработал специальный план эвакуации. Вместо того, чтобы каждому служащему индивидуально приписать, в какое бомбоубежище он должен бежать, для каждого муниципального здания определили, сколько служащих из него в какое бомбоубежище должны бежать. Задача индивидуального распределения была переложена на внутреннее управление муниципальных зданий.

План эвакуации учитывает количество служащих в каждом здании — каждый служащий должен быть учтен в плане и в каждое бомбоубежище может быть направлено количество служащих, не превосходящее вместимости бомбоубежища.

Городской Совет заявляет, что их план эвакуации оптимален в том смысле, что суммарное время эвакуации всех служащих города минимально.

Мэр города, находящийся в постоянной конфронтации с Городским Советом, не слишком то верит этому заявлению. Поэтому он нанял Вас в качестве независимого эксперта для проверки плана эвакуации. Ваша задача состоит в том, чтобы либо убедиться в оптимальности плана Городского Совета, либо доказать обратное, представив в качестве доказательства другой план эвакуации с меньшим суммарным временем для эвакуации всех служащих.

Карта города может быть представлена в виде квадратной сетки. Расположение муниципальных зданий и бомбоубежищ задается парой целых чисел, а время эвакуации из муниципального здания с координатами (X_i, Y_i) в бомбоубежище с координатами (P_j, Q_j) составляет $D_{ij} = |X_i - P_j| + |Y_i - Q_j| + 1$ минут.

Формат входного файла

Входной файл содержит описание карты города и плана эвакуации, предложенного Городским Советом. Первая строка входного файла содержит два целых числа N ($1 \leq N \leq 100$) и M ($1 \leq M \leq 100$), разделенных пробелом. N — число муниципальных зданий в городе (все они занумерованы числами от 1 до N), M — число бомбоубежищ (все они занумерованы числами от 1 до M).

Последующие N строк содержат описания муниципальных зданий. Каждая строка содержит целые числа X_i , Y_i и B_i , разделенные пробелами, где X_i , Y_i ($-1000 \leq X_i, Y_i \leq 1000$) — координаты здания, а B_i ($1 \leq B_i \leq 1000$) — число служащих в здании.

Описание бомбоубежищ содержится в последующих M строках. Каждая строка содержит целые числа P_j , Q_j и C_j , разделенные пробелами, где P_j , Q_j ($-1000 \leq P_j, Q_j \leq 1000$) — координаты бомбоубежища, а C_j ($1 \leq C_j \leq 1000$) — вместимость бомбоубежища.

В следующих N строках содержится описание плана эвакуации. Каждая строка представляет собой описание плана эвакуации для отдельного здания. План эвакуации из i -го здания состоит из M целых чисел E_{ij} , разделенных пробелами. E_{ij} ($0 \leq E_{ij} \leq 10000$) — количество служащих, которые должны эвакуироваться из i -го здания в j -е бомбоубежище.

Гарантируется, что план, заданный во входном файле, корректен.

Формат выходного файла

Если план эвакуации Городского Совета оптимален, то выведите одно слово **OPTIMAL**. В противном случае выведите на первой строке слово **SUBOPTIMAL**, а в последующих N строках выведите Ваш план эвакуации (более оптимальный) в том же формате, что и во входном файле. Ваш план не обязан быть оптимальным, но должен быть лучше плана Городского Совета.

Пример

stdin	stdout
3 4 -3 3 5 -2 2 6 2 2 5 -1 1 3 1 1 4 -2 -2 7 0 -1 3 3 1 1 0 0 0 6 0 0 3 0 2	SUBOPTIMAL 3 0 1 1 0 0 6 0 0 4 0 1
3 4 -3 3 5 -2 2 6 2 2 5 -1 1 3 1 1 4 -2 -2 7 0 -1 3 3 0 1 1 0 0 6 0 0 4 0 1	OPTIMAL

Разбор задачи G. План эвакуации

Построим двудольный граф. Вершинам первой доли соответствуют здания, второй — бомбоубежища. Из каждой вершины первой доли существует ребро в каждую вершину второй доли бесконечной пропускной способности и цены, равной расстоянию между зданием и бомбоубежищем. Добавим исток и сток. Проведем ребра из истока во все вершины первой доли, из всех вершин второй доли в сток нулевой стоимости и пропускной способности, равной соответствующему количеству людей. Оптимальному плану эвакуации соответствует максимальный поток минимальной стоимости.

Задача H. Два кратчайших пути

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Вчера Вася и Петя сильно повздорили и сегодня не хотят видеть друг друга у себя на пути в школу. Загвоздка в том, что они живут в одном доме, выходят в школу в одно и то же время и идут с одинаковой скоростью по кратчайшему пути. Никто из ребят не хочет менять свои привычки, и чтобы не идти вместе вдоль одной дороги, они хотят найти два разных кратчайших пути. Однако они могут встречаться в узлах дорожной сети, ибо кратковременная встреча не вызывает у них приступов раздражения. Ребята просят вас помочь им. Для этого они занумеровали узлы числами от 1 до N . Их дом и школа стоят в узлах с номерами 1 и N соответственно. Между парой узлов может быть не более одной дороги.

Формат входного файла

Первая строка содержит целые числа N и M ($2 \leq N \leq 400$), где M — число дорог, которое заметили Петя и Вася. Каждое из следующих M строк содержит по три целых числа: X , Y и L ($1 \leq X, Y \leq N$; $1 \leq L \leq 10000$), где X и Y — номера узлов, соединенных дорогой и длина дороги.

Формат выходного файла

В первую строку выведите номера узлов первого кратчайшего пути. Во вторую — второго. Если решения не существует, выведите “No solution” (без кавычек).

Пример

stdin	stdout
6 8	1 3 4 5 6
1 2 1	1 2 4 6
3 2 1	
3 4 1	
1 3 2	
4 2 2	
4 5 1	
5 6 1	
4 6 2	

Разбор задачи Н. Два кратчайших пути

Задача очень похожа на задачу Е при $k = 2$. После нахождения двух независимых путей наименьшей суммарной длины при помощи потока, нужно проверить, являются ли они оба кратчайшими.

Задача I. Остоз

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 с
 Ограничение по памяти: 64 Мб

Для излечения от почечного фиброкистозного остоза необходимого переправиться через болото. Для переправы можно использовать доски, которые соединяют кочки. После того, как на кочке кто-нибудь побывал, она тонет. Вам требуется излечить максимальное количество людей от почечного фиброкистозного остоза.

Формат входного файла

В первой строке входного файла записано число досок n ($1 \leq n \leq 1000$). Далее для каждой доски записаны координаты кочек — концов доски ($-2^{31} \leq x_i, y_i \leq 2^{31}$). Затем записаны координаты начальной и конечной точек (точки различны и доски, их соединяющей, нет). Все числа во входном файле целые.

Формат выходного файла

Выведите максимально количество людей, которых можно излечить.

Пример

stdin	stdout
8	2
0 0 1 0 1 0 2 1 1 0 2 -1	
2 1 3 0 2 -1 3 0 1 0 4 0	
3 0 4 0 0 0 3 0 0 0 4 0	

Разбор задачи I. Остоз

Необходимо найти в графе максимальное количество путей, не пересекающихся по вершинам. Раздвоим каждую вершину v на v_1 и v_2 . Проведем ориентированное ребро (v_1, v_2) . Каждое неориентированное ребро (u, v) исходного графа заменим на два ориентированных: (u_2, v_1) и (v_2, u_1) . Все ребра имеют пропускную способность 1. Заметим, что мы свели задачу о путях, не пересекающихся по вершинам, к задаче о путях, не пересекающихся по ребрам. Полученная задача решается используя максимальный поток.

Задача J. Поток в меняющейся сети

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 с
 Ограничение по памяти: 64 Мб

Задана ориентированная сеть, с которой производят два вида операций. Первая операция увеличивает на 1 пропускную способность ребра из x в y , а вторая операция уменьшает на 1 пропускную способность ребра из x в y .

Ваша задача выводить значение максимального потока в сети из вершины 1 в вершину n после каждой из операций.

Формат входного файла

В первой строке записаны целые числа n , m ($2 \leq n \leq 150$; $0 \leq m \leq 2000$), где n — это количество вершин в сети, а m — количество ребер. Далее в m строках перечислены сами ребра (x, y) с пропускной способностью c по одному в строке тройками чисел x, y, c ($1 \leq x, y \leq n$; $1 \leq c \leq 1000$). Далее входные данные содержат число t ($1 \leq t \leq 500$), где t — количество операций производимых над

сетью. Затем t строк содержат каждую из операций. Операции задаются последовательностями "1 x y " и "2 x y " в зависимости от типа.

Сеть в любой момент времени не содержит кратных ребер и петель. Пропускная способность любого ребра в сети в любой момент времени — неотрицательна.

Формат выходного файла

Выведите $t + 1$ число. В качестве первого числа выведите значение максимального потока в заданной сети до произведения всех операций. Далее выведите t чисел — значения максимального потока после каждой операции.

Примеры

stdin	stdout
4 5	15
1 2 15	16
2 3 5	15
3 4 15	
1 3 5	
2 4 5	
2	
1 2 3	
2 1 3	

Разбор задачи J. Поток в меняющейся сети

Находим максимальный поток алгоритмом Эдмондса-Карпа. Увеличение пропускной способности обрабатывается просто: нужно попытаться пустить дополнительную единицу потока. Представим, что ребро (u, v) полностью насыщено потоком, и мы хотим уменьшить его пропускную способность. Тогда в вершине u образуется избыток 1, а в вершине v — недостаток 1. Сделаем вершину u новым истоком в сети, вершину v — новым стоком и попытаемся пустить поток величины 1 в остаточной сети имеющегося потока. Если получилось — максимальный поток не изменился, если нет — уменьшился на 1. Осталось корректно перестроить поток, пустив единицу потока из u в исток и единицу потока из стока в v .

Задача К. Произвольная циркуляция

Вход: `stdin`
Выход: `stdout`
Ограничение по времени: 2 с
Ограничение по памяти: 64 Мб

Задана ориентированная сеть G . Для каждого ребра кроме его пропускной способности c_i , указана еще и нижнее ограничение на поток по ребру l_i . Это обозначает, что поток по ребру должен удовлетворять неравенству $l_i \leq f_i \leq c_i$.

Циркуляцией в сети называется любой поток величины 0. То есть для циркуляции верно, что в любую вершину сколько единиц потока втекает, столько и вытекает из нее (в случае потока это не верно для истока и стока сети).

Ваша задача найти произвольную циркуляцию в заданной сети.

Формат входного файла

В первой строке записана пара целых чисел ($1 \leq n \leq 200$; $0 \leq m \leq 400$), где n — количество вершин в сети, а m — количество ребер. Далее в m строках перечислены ребра в формате $from_i, to_i, l_i, c_i$, где $from_i$ это индекс стартовой вершины ребра, to_i это индекс конечной вершины ребра, l_i — нижняя граница величины потока по ребру, а c_i — верхняя граница ($1 \leq from_i, to_i \leq n$; $0 \leq l_i \leq c_i \leq 10^5$). Все числа — целые. В графе отсутствуют кратные ребра и петли.

Если в сети есть ребро из a в b , то в сети отсутствует ребро из b в a (т.е. встречных ребер нет).

Формат выходного файла

Если решения не существует, то выведите NO. В противном случае выведите YES. Далее в случае положительного ответа выведите m строк. m -я строка должна содержать количество потока, протекающего по i -ой дуге в искомой циркуляции.

Примеры

stdin	stdout
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 К. Произвольная циркуляция

Добавим две дополнительные вершины — исток и сток. Раздвоим каждое ребро графа на два ребра с пропускными способностями l_i и $r_i - c_i$. Первую половину разорвем: вместо него добавим ребро пропускной способности l_i из начальной вершины в сток и из истока — в конечную. Второе оставим без изменений. Построим в получившейся сети максимальный поток (алгоритмом Эдмондса-Карпа). Как построенный поток связан с циркуляцией в исходной сети? Если он насытил все ребра, выходящие из истока (и все ребра, входящие в сток), мы можем заменить ребра из истока и ребра в сток на ребра между вершинами исходного графа с нижними пропускными способностями l_i . Они будут насыщены, поэтому мы получим циркуляцию, удовлетворяющую условиям $l_i \leq f_i \leq c_i$. Если максимальный поток не насыщает все ребра из истока, задача не имеет решения.

Задача L. Сетевые войны

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 с
 Ограничение по памяти: 64 Мб

Компьютерная сеть Байтландии состоит из n компьютеров, соединенных m оптоволоконными кабелями. Каждый кабель соединяет какие-то

два компьютера и может передавать данные в обоих направлениях. Два компьютера сети особенно важны — один имеет выход в интернет, а другой соединен с президентской резиденцией.

Компьютер, соединенный с президентской резиденцией, имеет номер 1, а компьютер, имеющий выход в интернет, — номер n .

Недавно компания *Max Traffic* решила взять под контроль некоторые соединения, так чтобы иметь возможность просматривать данные, передаваемые пользователями из президентской резиденции. Конечно же *Max Traffic* хочет контролировать такой набор соединений, чтобы все данные, передаваемые из интернета в президентскую резиденцию, проходили хотя бы через одно из контролируемых соединений.

Чтобы притворить свой план в жизнь, компании необходимо купить соответствующие соединения у их текущих владельцев. Каждое соединение имеет определенную цену. Поскольку основная специализация компании не шпионаж, а предоставление интернет услуг, руководство компании хочет, чтобы средняя цена покупки была минимальна. Если компания покупает k соединений суммарной стоимостью c , то требуется минимизировать значение c/k .

Формат входного файла

Первая строка входного файла содержит целые числа n ($2 \leq n \leq 100$) и m ($1 \leq m \leq 400$). Последующие m строк содержат описания соединений — каждый кабель описывается тремя целыми числами: номерами компьютеров, которые он соединяет, и стоимостью покупки. Стоимость каждого соединения положительна и не превосходит 10^7 .

Любые два компьютера соединены не более, чем одним кабелем. Никакой компьютер не соединен сам с собой. Гарантируется, что сеть связна, т.е. данные можно передать от любого компьютера до любого другого по проводам сети.

Формат выходного файла

На первой строке выведите число соединений, которое надо купить. Во второй строке выведите номера этих соединений. Соединения нумеруются начиная с 1 в том же порядке, в каком они заданы во входном файле.

Пример

stdin	stdout
6 8 1 2 3 1 3 3 2 4 2 2 5 2 3 4 2 3 5 2 5 6 3 4 6 3	4 3 4 5 6
4 5 1 2 2 1 3 2 2 3 1 2 4 2 3 4 2	3 1 2 3

Разбор задачи L. Сетевые войны

Разрезом в сети $G = (V, E)$ будем называть множество ребер $C \subset E$, такое, что сток t недостижим из истока s по ребрам из $E \setminus C$. Задача состоит в нахождении разреза минимальной средней пропускной способности. О разрезах можно подробнее почитать в книге Т. Кормена и др. [1]. Построение минимального разреза (по суммарной пропускной способности) сводится к нахождению максимального потока. Чтобы минимизировать *среднюю* величину разреза, воспользуемся следующим приемом.

Предположим, что ответ на задачу равен w . Рассмотрим произвольный разрез. Пусть он состоит из k ребер с пропускными способностями c_1, c_2, \dots, c_k . Тогда

$$\frac{1}{k} \sum_{i=1}^k c_i \geq w.$$

Преобразуем это неравенство:

$$\sum_{i=1}^k (c_i - w) \geq 0.$$

Рассмотрим новую сеть, пропускные способности в которой равны $c(u, v) - w$. Величина минимального разреза (по суммарной величине) в

ней равна нулю. Будем подбирать значение w бинарным поиском. Ясно, что величина минимального разреза в сети с пропускными способностями $c(u, v) - w$ монотонна по w . Пропускные способности в такой сети могут быть отрицательными и не обязательно целые. Заметим, что все отрицательные ребра должны быть обязательно включены в минимальный разрез (каждое неориентированное ребро включается один раз). После этого удалим их из сети и будем искать максимальный поток в сети, в котором присутствуют только ребра с неотрицательной пропускной способностью. Например алгоритмом Эдмондса-Карпа.

Задача М. Снег в Берляндии

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Зимы в Берляндии снежные, и нынешняя зима не исключение. Каждую зиму правительство страны решает задачу чистки дорог от снега. И эта задача особенно сложна для столицы.

Можно считать, что столица Берляндии состоит из n перекрестков и m односторонних дорог. Каждая дорога соединяет два различных перекрестка x_i, y_i и направлена от x_i к y_i . На i -ой дороге лежит w_i тонн снега.

Для чистки дорог правительство наняло частную компанию “Snow White”. Каждый день компания высылает одну снегоуборочную машину. Она начинает работу на перекрестке с номером A , проходит несколько перекрестков до B -го и останавливается. Путь машины может содержать повторяющиеся дороги и перекрестки, включая A и B .

Грузовик за день проделывает путь от перекрестка A до перекрестка B , водитель, конечно, не нарушает правила дорожного движения. Каждый проход машины по дороге со снегом уменьшает количество снега на ней на 1 тонну. Так как жители столицы могут уличить правительство в бесцельном растрачивании бюджета, водителю запрещено вести машину по дороге, на которой нет снега.

Некоторые городские дороги представляют историческую ценность, потому что на них стоят правительственные здания. Поэтому по завершении работы компании “Snow White” эти дороги должны быть свободны от снега. Известно, на перекрестке A находится исторический центр столицы, что означает пешую досягаемость всех исторических дорог из центра по историческим дорогам. Помните, что с точки зрения пешехода дороги являются двусторонними.

Правительство платит компании за каждый день работы, поэтому руководство “Snow White” заинтересовано в том, чтобы растянуть работу на наибольшее количество дней.

Ваша задача — найти последовательность маршрутов из A в B , удовлетворяющих ограничениям: ограничение направления движения, чистота исторически значимых дорог, максимальное количество рабочих дней.

Формат входного файла

Первая строка содержит целые числа n, m, A, B ($2 \leq n \leq 100$; $0 \leq m \leq 5000$; $1 \leq A, B \leq n$; $A \neq B$), где n — число перекрестков в столице Берляндии, m — число дорог. Следующие m строк содержат четыре числа x_i, y_i, w_i, t_i ($1 \leq x_i, y_i \leq n$; $x_i \neq y_i$; $0 \leq w_i \leq 100$; $0 \leq t_i \leq 1$), где x_i, y_i — концы дороги, w_i — количество снега на ней, t_i — тип дороги (0 означает, что дорога обычная, 1 — историческая). Между парой перекрестков может быть не более одной дороги в каждом направлении.

Формат выходного файла

Выведите p — максимальное количество рабочих дней. Следующие p строк должны содержать описания ежедневных маршрутов снегоуборочной машины. Маршрут выводите как список номеров перекрестков, начинающийся с A , заканчивающийся в B , разделенных пробелом. Если есть несколько решений, выведите любое, если решение нет, выведите 0.

Примеры

stdin	stdout
4 7 1 4 1 2 3 1 2 1 100 0 2 4 1 0 1 3 1 0 3 4 4 0 2 3 2 1 1 4 2 0	6 1 3 4 1 4 1 4 1 2 4 1 2 3 4 1 2 3 4
3 3 1 2 1 3 2 0 3 2 3 0 1 2 1 0	3 1 3 2 1 3 2 1 2

Разбор задачи М. Снег в Берляндии

Построим сеть, в которой роль пропускных способностей играет количество снега w_i . Необходимо найти в этой сети максимальный поток, насыщающий заданное множество дуг V . Воспользуемся стандартным приемом: добавим две новые вершины — новый исток S и новый сток T , а также дугу из B в A бесконечной пропускной способности. “Разорвем” каждую дугу (u, v) из V : удалим (u, v) и добавим дуги (S, v) и (u, T) соответствующей пропускной способности $c(u, v)$. В новой сети найдем максимальный поток. В случае если этот поток не насыщает все дуги (S, v) и (u, T) , то, очевидно, задача не имеет решения. В противном случае найденному потоку соответствует некоторый поток в исходной сети, насыщающий все дуги из V . Однако он может не удовлетворять требованию максимальности.

Заметим, что величина потока в исходной сети равна потоку из B в A по добавленной дуге в перестроенной сети. Регулируя пропускную способность этой дуги, можно влиять на величину потока. А именно, пусть текущая величина потока — F_1 , а максимальная величина — F_2 . Тогда существует поток любой величины $F_1 \leq F \leq F_2$ и не существует потока большей величины. Поэтому для нахождения F_2 можно применить бинарный поиск в сочетании с уже описанным приемом “разрыва” дуг, используемого теперь и для дуги из B в A .

Другой способ построения максимального потока заключается в установлении отрицательных пропускных способностей для обратных дуг к дугам из V (чтобы пущенный по ним поток не мог “течь обратно”). После этого в исходной сети с некоторым уже построенным потоком, насыщающим дуги из V , нужно запустить алгоритм нахождения максимального потока, чтобы он “достроил” поток.

Отметим, что результирующий поток будет связным (имеется в виду, что он не будет содержать изолированных циклов) в силу специального условия в задаче о достижимости всех исторических дорог.

Осталось “выделить” требуемый набор маршрутов. Построим ориентированный мультиграф, в котором из u в v ведет столько же дуг, сколько потока течет из u в v в исходной сети. Кроме того, добавим дуги из B в A в количестве, равном построенному максимальному потоку. Нетрудно видеть, что мы получили эйлеров граф. Найдем в нем эйлеров цикл, “разорвем” его по фиктивным ребрам из B в A и получим требуемые пути.

О некоторых коварных случаях: при реализации нужно проявлять особую аккуратность с изначально имеющимися в сети дугами из A в B и из B в A .

День третий (18.02.2013 г.) Контеcт Алексея Шмелева

Об авторе...

Алексей Шмелев, закончил Нижегородский государственный университет.

Основные достижения:

- Победитель турнира ICL-2011 в составе команды ННГУ
- Серебряный призер финала ACM ICPC 2011 в составе команды ННГУ (5 место).
- Участник Google Code Jam 2011 onsite (20 место).
- Участник Russian Code Cup 2011 onsite (12 место).
- Участник Challenge 24 onsite в составе команды _NiN_ (8 место).
- Участник финала ACM ICPC 2012 в составе команды ННГУ (18 место).
- Рейтинг Codeforces: 2391.
- Рейтинг TopCoder: 2407.



Теоретический материал. Использование STL для решения олимпиадных задач (алгоритмы, контейнеры, примеры, ошибки)

В лекции будет рассказано про использование STL для решения олимпиадных задач. Цель лекции, прежде всего, показать наиболее часто используемые классы и алгоритмы STL.

Преимущества использования STL-алгоритмов:

- увеличение скорости написания программ;
- гарантированное отсутствие ошибок;
- универсальность (не зависят от типов данных);
- оптимальная асимптотика и использование памяти.

Недостатки:

- невозможность простой отладки;
- “непредсказуемое” поведение.
- “локальность” - сложность обращения к глобальным переменным.

Итератор

Объект, обладающий следующими операторами

++ - переходит на следующий элемент;

-- - переходит на предыдущий элемент;

* - разыменование, получение значения элемента.

Указатель - частный случай итератора.

Для алгоритмов STL пара итераторов задает множество объектов. В алгоритме будут рассмотрены все объекты от первого итератора (включительно) до второго (не включительно).

Random Access Iterator

Random Access Iterator - итератор, позволяющий переходить сразу на N элементов. Поддерживает операторы += и -=. Некоторые алгоритмы требуют передачи именно Random Access итераторов.

1. Алгоритмы.

```
1 #include <algorithm>
```

Большинство STL-алгоритмов позволяют находить какие-либо характеристики множества объектов одного типа, обычно задаваемого двумя итераторами.

В алгоритмах, каким-либо образом касающихся сравнения элементов (например, при сортировке) можно задать функцию сравнения элементов вида:

```
1 bool compare_func(T a, T b)
```

где T - тип объектов, для которых выполняется алгоритм. Функция должна возвращать true, если первый элемент должен быть меньше (идти раньше), чем второй. false - в противном случае. Если элементы равны, то функция должна возвращать false. Обычно функцию сравнения можно указать в качестве дополнительного (последнего) алгоритма.

Алгоритм count

```
1 count(it1, it2, value);
```

Считает количество вхождений элемента value в указанное множество, задаваемое итераторами it1, it2. Возвращаемое значение - int.

Пример:

```
1 a[5] = {1, 2, 8, 2, 2};
2 cout << count(a, a + 5, 2); // 3
3 cout << count(a, a + 5, 8); // 1
4 cout << count(a, a + 4, 2); // 2
```

Алгоритм find

```
1 find(it1, it2, value);
```

Находит итератор, соответствующий элементу, равному value в указанном множестве, задаваемом итераторами it1, it2. Возвращаемое значение - итератор.

Пример:

```
1 a[5] = {6, 2, 8, 1, 3};
2 find(a, a + 5, 2); // &a[1]
3 find(a, a + 5, 4); // &a[5]
4 find(a, a + 5, 1) - a; // = 3
```

Алгоритмы min_element/max_element

```
1 min_element(it1, it2);
2 max_element(it1, it2);
```

Находит итератор, соответствующий минимальному/максимальному элементу, в указанном множестве, задаваемом итераторами it1, it2. Возвращаемое значение - итератор.

Пример:

```
1 a[5] = {6, 2, 8, 1, 3};
2 min_element(a, a + 5); // &a[3]
3 max_element(a, a + 5); // &a[2]
4 min_element(a, a + 5) - a; // = 3
5 max_element(a, a + 5) - a; // = 2
6 *min_element(a, a + 5); // = 1
7 *max_element(a, a + 5); // = 8
```

Алгоритм binary_search

```
1 binary_search(it1, it2, value);
```

Проверяет, находится ли элемент `value` в указанном множестве, задаваемом итераторами `it1`, `it2`. Множество должно быть отсортировано. Возвращаемое значение - `true`, если элемент найден, `false` - в противном случае.

Пример:

```
1 a[5] = {1, 2, 8, 17, 239};
2 cout << binary_search(a, a + 5, 17); // true
3 cout << binary_search(a, a + 5, 19); // false
```

Алгоритмы lower_bound/upper_bound

```
1 lower_bound(it1, it2, value);
2 upper_bound(it1, it2, value);
```

Возвращает итератор, соответствующий первому/последнему элементу на множестве `[it1, it2)`, перед которым можно вставить элемент `value`, сохраняя отсортированность множества.

Возвращаемое значение - итератор.

Пример:

```
1 a[5] = {1, 2, 2, 6, 8};
2 lower_bound(a, a + 5, 1); // &a[0]
3 upper_bound(a, a + 5, 1); // &a[1]
4 lower_bound(a, a + 5, 2); // &a[1]
5 upper_bound(a, a + 5, 2); // &a[3]
```

Алгоритм sort

```
1 sort(it1, it2);
```

Сортирует множество между итераторами `it1`, `it2`.

Пример:

```
1 a[5] = {6, 2, 8, 1, 3};
2 sort(a, a + 5); // a[5] = {1, 2, 3, 6, 8}
```

Алгоритм stable_sort

Аналогичен алгоритму `sort`, но сохраняет относительный порядок элементов.

Алгоритм replace

```
1 replace(it1, it2, value1, value2);
```

Заменяет все вхождения `value1` на `value2` в указанном множестве, задаваемым итераторами `it1`, `it2`.

Пример:

```
1 a[5] = {1, 2, 2, 6, 8};
2 replace(a, a + 5, 2, 4); // a[5] = {1, 4, 4, 6, 8}
```

Алгоритм reverse

```
1 reverse(it1, it2);
```

Меняет порядок элементов между итераторами it1 и it2 на противоположный.

Пример:

```
1 a[5] = {6, 2, 8, 1, 3};
2 sort(a, a + 5); // a[5] = {1, 2, 3, 6, 8}
```

Алгоритм stable_sort

Аналогичен алгоритму sort, но сохраняет относительный порядок элементов.

Алгоритм replace

```
1 replace(it1, it2, value1, value2);
```

Заменяет все вхождения value1 на value2 в указанном множестве, задаваемым итераторами it1, it2.

Пример:

```
1 a[5] = {1, 2, 2, 6, 8};
2 replace(a, a + 5, 2, 4); // a[5] = {1, 4, 4, 6, 8}
```

Алгоритм reverse

```
1 reverse(it1, it2);
```

Меняет порядок элементов между итераторами it1 и it2 на противоположный.

Пример:

```
1 a[5] = {6, 2, 8, 1, 3};
2 reverse(a, a + 5); // a[5] = {3, 1, 8, 2, 6}
```

Алгоритм merge

```
1 merge(it11, it12, it21, it22, out_it)
```

Сливает два отсортированных множества, задаваемых итераторами it11, it12 (первое множество) и итераторами it21, it22 (второе множество), копируя элементы в новое множество, начиная с итератора out_it. Возвращаемое значение - итератор, соответствующий концу получившегося множества.

Пример:

```
1 a[5] = {1, 2, 2, 6, 8};
2 b[3] = {2, 3, 9};
3 *c;
4 merge(a, a + 5, b, b + 3, c); // возвращает &c[8], c[8] = {1, 2, 2, 2, 3, 6, 8, 9}
```

Алгоритм unique

```
1 unique(it1, it2);
```

“Сжимает” отсортированное множество элементов, задаваемое итераторами it1 и it2, удаляя одинаковые элементы. Возвращаемое значение - итератор, соответствующий концу “сжатого” множества.

Пример:

```
1 a[5] = {1, 2, 2, 6, 6};
2 unique(a, a + 5); // возвращает &a[3], a[3] = {1, 2, 6}
```

Алгоритм random_shuffle

```
1 random_shuffle(it1, it2)
```

Случайным образом перемешивает элементы множества, задаваемого итераторами it1 и it2.

Пример:

```
1 a[5] = {6, 5, 1, 4, 3};
2 random_shuffle(a, a + 5); // возможный результат a[5] = {1, 5, 6, 3, 4};
```

Алгоритм next_permutation

```
1 next_permutation(it1, it2)
```

Перемещает элементы множества, задаваемого итераторами it1 и it2 таким образом, чтобы они образовали лексикографически следующую последовательность. Возвращает true, если последовательность нашлась и множество изменено, false - если множество уже образовало лексикографически максимальную последовательность (в этом случае множество не изменяется).

Пример:

```
1 a[5] = {6, 1, 1, 5, 3};
2 next_permutation(a, a + 5); // возвращает true, a[5] = {6, 1, 3, 1, 5}
3 a[5] = {6, 5, 3, 1, 1};
4 next_permutation(a, a + 5); // возвращает false, a[5] = {6, 5, 3, 1, 1}
```

Алгоритм prev_permutation

Аналогичен next_permutation, но задает на множестве лексикографически предыдущую перестановку.

2. Класс pair.

Позволяет работать сразу с парой элементов произвольных (не обязательно одинаковых типов).

Объявление:

```
1 pair<type1, type2> var;
```

Содержит 2 поля: `var.first` - первый элемент пары `var.second` - второй элемент пары.

Преимущества:

- универсальность (подходит для любых классов);
- передача одного параметра вместо двух в функциях;
- стандартный компаратор: сравнение сначала по первому элементу, в случае равенства - по второму;
- функция `make_pair`, позволяющая не указывать типы элементов в паре.

Примеры:

```
1 int x, y;
2 double w;
3 ...
4 pair<int, int> coordinates = pair<int, int> (x, y);
5 pair<int, int> coordinates = make_pair(x, y);
6 pair<double, pair<int, int> > point = make_pair(w, make_pair(x, y));
```

3. Контейнеры.

Общие свойства

Контейнеры позволяют хранить в себе множества элементов одного типа и выполнять какие-либо операции над ними, итерировать и т.п. При создании контейнера необходимо указывать, какого типа элементы он будет в себе хранить.

Методы:

- `size()` - возвращает размер контейнера (количество элементов в нем);
- `empty()` - проверяет, является ли контейнер пустым;
- `clear()` - очищает контейнер;
- методы добавления элементов в контейнер (различаются между разными контейнерами);
- методы удаления элементов;
- итераторы.

Контейнер `vector`

“Динамический массив”, хранит добавляемые элементы в виде массива (в порядке добавления их в контейнер). В отличие от статического массива, позволяет добавлять произвольное число элементов, не задавая размер

заранее. При добавлении элементов если уже выделенной памяти не достаточно, то выделяется память вдвое большего размера, чем текущий.

```
1 #include <vector>
```

Методы:

- `push_back(value)` - добавить элемент `value` в конец вектора;
- `pop_back()` - удалить последний элемент из вектора;
- `capacity()` - текущий размер выделенной памяти;
- `resize(size)` - изменить размер до заданного значения (`size`);
- `reserve(cap)` - зарезервировать в векторе указанное число элементов (с учетом уже имеющихся). Меняется значение `capacity()`.

Итераторы:

- `begin()` - итератор, соответствующий первому элементу;
- `end()` - итератор, соответствующий позиции после последнего элемента.

Примеры:

```
1 vector<int> v;  
2 v.push_back(100); // вектор содержит один элемент - 100  
3 v.push_back(500); // вектор содержит два элемента - 100 и 500  
4 *v.begin(); // 100  
5 v.size(); // 2
```

Использование

Обход вектора через индекс элемента:

```
1 vector<int> v;  
2 ...  
3  
4 for (int i = 0; i < v.size(); i++) {  
5     int x = v[i];  
6     ...  
7 }
```

Обход вектора через итераторы:

```
1 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
2     int x = *it;  
3     ...  
4  
5 }
```

Применение алгоритмов к множеству элементов вектора.

```
1 vector<long long> v;
2 ...
3 sort(v.begin(), v.end());
4 int c = count(v.begin(), v.end(), 10011);
```

Использование в задачах

1. Дан граф из N вершин и M ребер. Необходимо сохранить список смежных вершин для каждой вершины.

```
1 vector<int> v[MAXN];
2 ...
3
4 cin >> n;
5 for (int i = 0; i < M; i++) {
6     int x, y;
7     cin >> x >> y;
8     v[x].push_back(y);
9     v[y].push_back(x);
10 }
```

2. “Отложенный” вывод. Нужно сохранить некоторые значения, после чего вывести их все в обратном порядке.

```
1 vector<int> ans;
2
3 ...
4 ans.push_back(value);
5 ...
6 reverse(all(ans));
7 for (int i = 0; i < ans.size(); i++)
8     cout << ans[i] << endl;
```

3. Временное хранилище. Нужно посчитать сумму высот K наименее высоких поддеревьев текущей вершины.

```
1 int f(int x) {
2     ...
3     vector<int> tmp;
4     ...
5     tmp.push_back(h[y]);
6     ...
7     sort(tmp.begin(), tmp.end());
8     int sum = 0;
9     for (int i = 0; i < min(k, tmp.size()); i++)
10         sum += tmp[i];
11     return sum;
12 }
```

4. Простые преобразования множества элементов. Выбрать уникальные элементы из множества.


```
1 vector<int> v;
2 ...
3 v.push_back(x);
4 ...
5
6 sort(v.begin(), v.end());
7 v.resize(unique(v.begin(), v.end()) - v.begin());
8 ...
```

Контейнер queue

Очередь, позволяет добавлять элементы в конец очереди, удалять элементы из начала очереди.

```
1 #include <queue>
```

Методы:

- `front()` - первый элемент;
- `pop()` - удаление первого элемента.
- `push(value)` - добавление элемента `value` в конец очереди.

Использование в задачах

1. Поиск в ширину

```
1 queue<pair<int, int> > q;
2
3 void process(pair<int, int> p) {
4     int x = p.first;
5     int y = p.second;
6     ...
7 }
8
9 ...
10
11 int main() {
12     ...
13     int x, y;
14     q.push(mp(x, y));
15     while (!q.empty()) {
16         process(q.front());
17         q.pop();
18     }
19     ...
20 }
```

Контейнер `priority_queue`

Очередь с приоритетами, позволяет добавлять элементы в очередь, извлекать первый (наибольший) элемент из очереди.

```
1 #include <queue>
```

Методы:

- `top()` - первый (наибольший) элемент;
- `pop()` - удаление первого элемента;
- `push(value)` - добавление элемента `value` в конец очереди.

Использование в задачах

1. Алгоритм Дейкстры

```
1 vector<pair<int, int> > v[MAXN];
2 int ans[MAXN];
3
4 priority_queue<pair<int, int> > q;
5
6 void process(pair<int, int> p) {
7     int d = -p.first;
8     int x = p.second;
9
10    if (ans[x] != d)
11        return;
12
13    for (int i = 0; i < v[x].size(); i++) {
14        int y = v[x][i].first;
15        int nd = d + v[x][i].second;
16
17        if (ans[y] > nd) {
18            q.push(make_pair(-nd, y));
19            ans[y] = nd;
20        }
21    }
22 }
23
24 int main() {
25     ...
26     int x;
27     q.push(mp(0, x));
28     while (!q.empty()) {
29         process(q.top());
30         q.pop();
31     }
```

Контейнер set

Множество, не добавляет одинаковые элементы. Операции добавления, поиска, удаления работают за $O(\log(N))$, где N - текущий размер множества.

```
1 #include <set>
```

Методы:

- `insert(value)` - добавляет элемент `value` в множество;
- `erase(value)` - удаляет элемент `value` из множества, если такого элемента не было в множестве, то не выполняет никаких действий;
- `erase(it)` - удаляет элемент из множества, соответствующий итератору `it`;
- `find(value)` - возвращает итератор на элемент `value` в множестве, если такого элемента нет, то возвращает `end()`;
- `lower_bound(value)` - аналогичен `lower_bound` для итераторов `begin()`, `end()`, работает за $O(\log(N))$;
- `upper_bound(value)` - аналогичен `upper_bound` для итераторов `begin()`, `end()`, работает за $O(\log(N))$.

Итераторы:

- `begin()` - итератор, соответствующий началу множества;
- `end()` - итератор, соответствующий концу множества.

Использование в задачах

1. `set` как online-множество уникальных элементов. На плоскости добавляются по одной N точек, нужно указать, сколько среди них различных после каждого шага;
2. `set` как динамическое множество. Задача построения наидлиннейшей последовательности, не содержащей одинаковых соседних элементов;
3. `set` как связный список. Задача - раскраска отрезка. Даны N запросов на перекрашивание подотрезка в определенный цвет. Необходимо определить итоговые цвета клеток отрезка;
4. `set` как очередь. Алгоритм Дейкстры.

Ошибки

1. Нетранзитивный компаратор.
2. Компаратор не различает различные элементы.
3. Нарушение относительного порядка элементов.
4. “Позднее” удаление из `set`.
5. Использование глобального метода `lower_bound` вместо метода контейнера (`lower_bound(s.begin(), s.end(), x)`).

Контейнер *multiset*

Аналогичен контейнеру `set`, но может хранить одинаковые элементы. При этом: метод `erase(value)` удаляет все значения `value` из сета. Метод `erase(it)` удаляет только указанный итератор.

Использование в задачах

1. Поиск максимума на отрезках заданной длины. Дано N чисел для каждого множества M подряд идущих чисел, нужно найти наибольшее.

Список литературы

1. Удобный справочник по C++, в том числе по STL
<http://www.cplusplus.com/reference/>
2. Статья на `topcoder.com`
<http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=standardTemplateLibrary>
<http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=standardTemplateLibrary2>

Задачи и разборы

Задача A. ДОМА 2: Last Hit

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

- Вася играет в популярную онлайн-игру *Defence of Mysterious Accepted*
2. В этой игре Васе нужно управлять героем, зарабатывать опыт и деньги,

уничтожать врагов, решать судьбу мира. В данной задаче Вы поможете Васе заработать больше денег, чтобы он смог купить полезные вещи. Наиболее простой способ заработать деньги — убить крипа (нейтрального юнита). Однако, Вася получит деньги только в том случае, если его герой нанес завершающий удар, в результате которого крип умер. Вася увидел крипа и решил его убить, однако, крип подошел близко к одной из оборонительных башен и она тоже стала атаковать крипа. Если последний удар крипу нанесет башня, а не Вася, то Вася не получит денег за этого крипа. Это недопустимо!

Как герой Васи, так и башня, обладают определенной скоростью атаки, то есть Вася не может наносить удар крипу чаще, чем раз в T_1 секунд, а башня — в T_2 секунд. Более формально, если Вася нанес удар в момент T , то следующий удар он может нанести не раньше, чем в момент времени $T + T_1$. Аналогичное условие выполняется и для башни. Для простоты будем считать, что Вася атакует крипа только в моменты времени, соответствующие целому числу секунд. Кроме того, герой Васи давно не наносил удары, то есть может сразу атаковать крипа (в момент времени 0). Башня же наносит первый удар крипу в момент времени 0 и далее атакует с максимальной скоростью, то есть наносит удар каждые T_2 секунд. Если Вася и башня нанесли удар в один и тот же момент времени, то считается, что сначала урон нанесла именно башня (см. пример 1).

Крип обладает здоровьем в H единиц. За каждый удар героя Васи у крипа отнимается H_1 единиц здоровья, за каждый удар башни — H_2 единиц. Если после очередного удара количество единиц здоровья крипа стало неположительным, то крип умирает.

Помогите Васе убить крипа и получить заслуженную награду! Для этого Вам нужно определить, в какие моменты времени герой Васи должен атаковать крипа.

Формат входного файла

В первой строке содержатся 2 целых числа — T_1 и T_2 ($1 \leq T_1, T_2 \leq 100000$) — минимальные интервалы между атаками героя Васи и башни соответственно. На следующей строке даны 3 целых числа — H_1 , H_2 и H ($1 \leq H_1, H_2, H \leq 100000$) — урон героя Васи, урон башни и количество единиц здоровья крипа.

Формат выходного файла

На первой строке вывести единственное число — количество ударов, которые должен нанести герой Васи, чтобы убить крипа. В следующей строке нужно перечислить времена нанесения ударов в порядке возрастания (между двумя соседними ударами Васи должно пройти хотя бы T_1

секунд). Если существует несколько решений, можно вывести любое из них.

Если Вася не может убить крипа ни при каких действиях — вывести -1.

Примеры

stdin	stdout
1 1 1 1 1	-1
1 1 1 1 2	1 0
1 2 1 4 10	2 2 3

Разбор задачи A. DOMA 2: Last Hit

Переберем, сколько ударов нанесет крипу башня, прежде чем мы уничтожим его. Поскольку $H \leq 100000$, то башня сделает не более 99999 ударов. Пусть мы зафиксировали число ударов башни - K_2 и хотим убить крипа, прежде чем башня нанесет следующий удар. Тогда прошло времени $(K_2 - 1) \cdot T_2$, крип получил $D_2 \cdot K_2$ урона. У него осталось здоровья $H' = H - D_2 \cdot K_2$. Необходимо, чтобы крип еще не умер, то есть $H' \geq 0$. До следующего удара башни пройдет еще $T_2 - 1$ секунд, т.е. $T = K_2 \cdot T_2 - 1$ (считая от начала). За это время мы должны нанести крипу достаточно урона, чтобы он умер, то есть хотя бы $K_1 = H' / D_1$ ударов. Для этого нам потребуется $T' = (K_1 - 1) \cdot T_1$ времени. Если $T' \leq T$, то мы можем нанести завершающий удар крипу. Во избежание “преждевременной” смерти, рекомендуется наносить удары как можно позднее, то есть в моменты времени $T, T - T_1, T - 2 \cdot T_1$ и т.д. Если ни для какого K_2 условие не выполнилось, то ответ на задачу -1.

Задача B. Last Effect 3: Danger

Вход: stdin
Выход: stdout
Ограничение по времени: 1 с
Ограничение по памяти: 256 Мб

Вася играет в Last Effect 3. Иногда он слишком заигрывается, теряет концентрацию и не замечает, что галактика уже в опасности. В данной задаче Вы должны помочь Васе защитить галактику от нападения пришельцев. На самом деле, Васе не очень-то и нужна вся галактика, но он

обязательно должен защитить 3 чрезвычайно важные планеты. Будем считать, что каждая планета представляет собой окружность на плоскости. Планеты могут иметь разные радиусы. Для защиты планеты Вася должен долететь до ее поверхности. Звездолет Васи поврежден и может двигаться только по прямой, поэтому для избежания несчастных случаев при посадке на планету, Вася должен двигаться по касательной к соответствующей окружности. Поскольку Васе важны все 3 планеты, то он хочет разместить свой звездолет в такой точке, что расстояния по касательной до каждой из этих планет были бы равны между собой. Помогите Васе найти такую точку на плоскости.

Формат входного файла

Входной файл содержит 3 строки. Каждая строка содержит 3 целых числа — координаты соответствующей окружности X_i, Y_i ($-100000 \leq X_i, Y_i \leq 100000$) и радиус окружности R_i ($1 \leq R_i \leq 100000$). Окружности не пересекаются и не касаются. Центры окружностей не лежат на одной прямой.

Формат выходного файла

Вывести пару вещественных чисел с абсолютной или относительной погрешностью не более 10^{-6} — координаты искомой точки. Гарантируется, что решение существует. Если решений несколько, то можно вывести любое из них.

Примеры

stdin	stdout
0 0 1 0 4 1 8 0 1	4.00000000 2.00000000

Разбор задачи В. Last Effect 3: Danger

Квадрат длины касательной от точки до окружности называется степенью точки относительно окружности и равен $D^2 - R^2$, где D - расстояние от точки до центра окружности, R - радиус окружности. Решим задачу для двух окружностей (X_1, Y_1, R_1) , (X_2, Y_2, R_2) . Найдем множество точек (X, Y) , для которых степени относительно двух окружностей равны. Получаем $(X - X_1)^2 + (Y - Y_1)^2 + R_1^2 = (X - X_2)^2 + (Y - Y_2)^2 + R_2^2$. Раскрыв скобки, мы получим линейное уравнение вида $A \cdot X + B \cdot Y + C = 0$, т.е. уравнение прямой (эта прямая называется радикальной осью двух окружностей).

Найдем аналогичную прямую для окружностей (X_2, Y_2, R_2) , (X_3, Y_3, R_3) и пересечем ее с первой прямой. Очевидно, что точка пересечения этих двух радикальных осей будет удовлетворять условию задачи (эта точка называется радикальным центром трех окружностей).

Задача C. Fineage: Training

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Вася играет в Fineage. В этой игре считается особо почетным стать эльфом 80-го уровня (правда, говорят, уже можно стать и эльфом 99-го уровня). Вася уже стал эльфом, осталось лишь получить хотя бы 80-й уровень. Для этого Вася тренирует своего персонажа каждый день, убивая генерированных вепрей в лесу. А именно, за каждый будний день Вася получает X_1 единиц опыта, за каждый выходной — X_2 единиц опыта. Вася живет по своему уникальному календарю. В его неделе ровно N дней, причем только один из них выходной. Правда, периодичность выходных Вася сохранил, то есть между двумя соседними выходными находится ровно $N - 1$ будний день.

Со временем Вася стал замечать, что часто видит одинаковые числа на экране в графе, показывающей количество опыта после очередного дня, а именно — его очень заинтересовала последняя цифра. В игре Fineage используется система счисления с основанием M , таким образом, последняя цифра может иметь значение от 0 до $M - 1$. Помогите Васе определить, каков же период у последней цифры значения опыта? Более формально, Вам нужно найти наименьшее значение $T > 0$, такое что для любого i выполняется $E_i = E_{i+T} \pmod{M}$, где E_i — суммарное количество опыта, которое Вася получил за первые i дней.

Формат входного файла

В первой строке дано 2 целых числа X_1 и X_2 — количество опыта, которое получает Вася в будний день и в выходной день соответственно ($0 \leq X_1, X_2 \leq 10^9$). Во второй строке находится единственное число N — количество дней в неделе Васи ($1 \leq N \leq 10^9$). В третьей строке находится число M — основание системы счисления в Fineage ($1 \leq M \leq 10^9$).

Формат выходного файла

Вывести единственное число — длину периода последней цифры в ко-

личестве опыта Васи.

Примеры

stdin	stdout
1 2	2
2	
3	

Разбор задачи C. Fineage: Training

Возьмем вместо X_1 и X_2 их остатки по модулю M . Если мы получили $X_1 = X_2$, то можно считать, что неделя состоит из одного дня. Пусть искомый период T , тогда легко проверить, что T должно делиться без остатка на N . Пусть $T = N \cdot K$. За N подряд идущих дней мы получим $X = X_1 \cdot (N - 1) + X_2$ опыта. Тогда за T дней мы получим $K \cdot X$ опыта. Значит, $K \cdot X$ делится на M . Тогда $K = M / \gcd(M, X)$, $K \cdot N$ - ответ на задачу.

Задача D. ДОМА 2: Inventory

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Вася играет в популярную онлайн-игру Defence of Mysterious Accepted 2. В этой игре нужно управлять героем, зарабатывать опыт и деньги, убивать врагов, решать судьбу мира. Еще здесь можно покупать предметы. Некоторые предметы являются “сложными”, то есть состоят из нескольких других предметов, остальные предметы назовем “простыми”. Некоторые части сложных предметов тоже могут быть сложными предметами. Например, предмет BattleFury состоит из простых предметов Claymore и Broadsword и сложного предмета Perseverance. Perseverance, в свою очередь, состоит из простых предметов RingOfHealth и VoidStone. Будем считать (хотя, в игре это не всегда так), что сложные предметы можно разбирать на составные части, то есть из сложного предмета можно получить множество простых предметов, из которых он состоит.

На поле битвы встретились два героя, каждый со своим набором предметов в инвентаре. Вам необходимо определить, какой из героев сильнее, то есть проверить, можно ли предметы одного набора разобрать на простые предметы, из которых можно собрать другой набор.

Формат входного файла

В первой строке находится единственное целое число N ($1 \leq N \leq 100$) — количество предметов в игре. Далее следуют N строк, в которых описаны предметы. Если предмет простой, то описание содержит только название этого предмета. Если предмет сложный, то после имени идет перечисление составных частей предмета в следующем формате:

$\langle \text{item} \rangle = \langle \text{part1} \rangle + \langle \text{part2} \rangle + \dots + \langle \text{partK} \rangle$

Где $\langle \text{item} \rangle$ — название сложного предмета, $\langle \text{part1} \rangle, \dots, \langle \text{partK} \rangle$ — название предметов, из которых состоит $\langle \text{item} \rangle$. Количество составных частей одного предмета не менее 2 и не превосходит 10. Символы $=$ и $+$ отделены ровно одним пробелом с каждой стороны. Название предмета — непустая строка, состоящая не более чем из 50 латинских символов, регистр учитывается. Названия различных предметов различны. Каждый предмет описан ровно один раз. Гарантируется, что для любого сложного предмета все его части так же описаны (не обязательно перед ним). Гарантируется, что в описании предметов нет циклических зависимостей, то есть каждый сложный предмет можно разобрать на конечное количество простых предметов (единственным образом).

В следующей строке находится целое число M ($1 \leq M \leq 100$) — количество предметов в инвентаре первого героя. В следующих M строках находятся названия этих предметов (по одному в строке). Инвентарь может содержать одинаковые предметы. Далее находится описание инвентаря второго героя в том же формате.

Формат выходного файла

Если предметы первого героя можно разобрать на части, из которых можно полностью сложить инвентарь второго героя (возможно, какие-то части останутся лишними), то необходимо вывести число 1. Если из предметов второго героя можно сложить инвентарь первого, то необходимо вывести число 2. Если оба условия выполнены, то число 0. Если ни одно условие не выполнено, то число -1.

Примеры

stdin	stdout
<pre>6 BattleFury = Claymore + Broadsword + Perseverance Claymore Broadsword Perseverance = RingOfHealth + VoidStone RingOfHealth VoidStone 1 BattleFury 1 Perseverance</pre>	1
<pre>6 BattleFury = Claymore + Broadsword + Perseverance Claymore Broadsword Perseverance = RingOfHealth + VoidStone RingOfHealth VoidStone 1 RingOfHealth 1 Perseverance</pre>	2
<pre>6 BattleFury = Claymore + Broadsword + Perseverance Claymore Broadsword Perseverance = RingOfHealth + VoidStone RingOfHealth VoidStone 2 Claymore Broadsword 2 RingOfHealth VoidStone</pre>	-1
<pre>6 BattleFury = Claymore + Broadsword + Perseverance Claymore Broadsword Perseverance = RingOfHealth + VoidStone RingOfHealth VoidStone 1 BattleFury 4 RingOfHealth VoidStone Claymore Broadsword</pre>	0

Разбор задачи D. DOMA 2: Inventory

Построим ориентированный граф, соответствующий предметам игры и сделаем топологическую сортировку предметов. Далее будем обрабатывать инвентарь первого героя в топологическом порядке, начиная с “самых сложных”. Будем разбирать предметы на составные части, при этом для любого предмета будем поддерживать количество - сколько раз он содержится в инвентаре. При “разбиении” очередного предмета обновим эти количества для составляющих частей. Будем продолжать эту операцию, пока не разберем весь инвентарь на простые предметы. Аналогичную процедуру проделаем для второго инвентаря. Далее сравним количества простых предметов у героев и получим ответ. Поскольку количество простых предметов после разбирания сложных может быть порядка 10^{100} , то для решения понадобится использовать длинную арифметику.

Задача E. South Mark: 3.50

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

CHEF'S DAD: Ooh, it must've been about seven, eight years ago. Me and the little lady was out on this boat, you see, all alone at night, when all of a sudden this huge creature, this giant crustacean from the paleolithic era, comes out of the water.

CHEF'S MOM: We was so scared, Lord have mercy, I jumped up in the boat and I said "Thomas, what on earth is that creature?!"

CHEF'S DAD: It stood above us looking down with these big red eyes,-

CHEF'S MOM: Oh, it was so scary!

CHEF'S DAD: -and I yelled. I said, "What do you want from us, monster?!" And the monster bent down and said, "...Uh I need about tree-fifty."

KYLE: What's tree-fifty?

CHEF'S DAD: Three dollars and fifty cents.

CHEF'S MOM: Tree-fifty.

Вася играет... и параллельно смотрит популярный сериал South Mark. В этот раз ему попала серия, где лох-несское чудовище постоянно требовало 3 доллара 50 центов. Вася задумался, а сможет ли он спастись от чудовища, дав ему 3.50, если попадет в такую же ситуацию? И сколько раз он сможет избежать гнева лох-несского друга? Более того, вдруг чудовище захочет не 3.50 центов, а N центов? Вася вспомнил, какие у него есть монеты, определил число N и попросил Вас посчитать, какое наибольшее число наборов, суммарным достоинством ровно N центов, можно составить из имеющегося множества монет.

Формат входного файла

В первой строке дано число N — предполагаемая сумма, которую будет требовать лох-несское чудовище ($1 \leq N \leq 10^9$). Во второй строчке дано число M — количество различных номиналов монет Васи ($1 \leq M \leq 30$). В каждой из следующих M строк записано два целых числа N_i — номинал соответствующей монеты в центах ($1 \leq N_i \leq 10^9$) и K_i — количество монет данного номинала у Васи ($1 \leq K_i \leq 10^9$). Известно, что для любого $i > 1$ N_i делится без остатка на N_{i-1} и $N_i > N_{i-1}$.

Формат выходного файла

Вывести единственное число — наибольшее количество наборов достоинством N , которое можно составить из данного множества монет.

Примеры

stdin	stdout
350	2
3	
10 10	
50 8	
100 4	

Разбор задачи E. South Mark: 3.50

Поскольку каждое следующее достоинство монеты делится на предыдущее, то мы можем использовать жадный алгоритм при составлении наборов суммой N . То есть, если мы хотим составить набор и нам нужно добавить монеты суммарным достоинством M , то выгодно сначала добавлять монеты наибольшего достоинства, не превосходящего M .

Будем искать ответ бинарным поиском. Пусть мы зафиксировали число желаемых наборов - K . Будем жадно расставлять монеты в наборах, начиная с монет с наибольшим достоинством.

На каждом шаге мы рассматриваем монеты некоторого достоинства X , которых имеется Y штук. Всего мы в каждый набор можем добавить $T = \min(N'/X, Y/K)$ монет, где N' - текущая сумма, которую нужно добавить к наборам. Если мы добавили Y/K монет в каждый набор, и у нас остались еще монеты (то есть Y не делится на K), то добавим оставшиеся монеты по одной в $Y \% K$ наборов, запишем $T = (Y/K) + 1$, и $D = (K - Y \% K) * X$ - "долг", который нам нужно компенсировать монетами меньшего достоинства (заметим, что D делится на любое следующее рассматриваемое достоинство монеты). После этого уменьшим N' на величину $T * X$ и перейдем к следующему номиналу монет. Если после просмотра всех монет мы получили, что $D > 0$ или $N' > 0$, то мы не можем собрать K наборов, иначе K наборов собрать можно.

Задача F. HOLM 2: The Great Battle

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вася играет в Heroes of Light and Magic 2. Одной из ключевых составляющих игры является проведение боя между двумя армиями. Каждая армия состоит из нескольких отрядов (стеков) одинаковых юнитов. Каждый юнит характеризуется двумя значениями: количеством единиц здоровья (H_i) и уровнем атаки, представляющим собой интервал ($L_i - R_i$). За одну атаку юнит может нанести урон от L_i до R_i включительно, конкретное значение нанесенного урона выбирается в игре случайным образом каждый раз, когда юнит атакует.

Вася управляет одной армией, а его противник — другой. За один ход игрок отдает команду одному из своих отрядов атаковать отряд противника. При этом для каждого юнита в отряде выбирается значение нанесенного урона (в интервале от L_i до R_i). Урон, нанесенный отрядом, равен сумме выбранных значений. В результате юниты отряда противника могут быть уничтожены. Происходит это следующим образом. Будем считать, что в каждом отряде задан порядок юнитов, в котором они будут получать урон. Если нанесенный отряду урон не меньше, чем текущее количество единиц жизни первого юнита отряда, то юнит умирает, а урон уменьшается на количество единиц жизни юнита. После чего аналогичная ситуация происходит со следующими юнитами в отряде. Если в какой-то момент количество единиц жизни текущего юнита больше, чем оставшийся урон, то здоровье юнита уменьшается на величину урона, а урон другим юни-

там больше не наносится. Таким образом, в каждом отряде максимум один юнит обладает не полным запасом здоровья. Отряд считается уничтоженным, если уничтожены все его юниты. Армия считается уничтоженной, если уничтожены все ее отряды. В данной задаче мы будем считать, что игроки могут ходить в произвольном порядке. Юниты не восстанавливают здоровье между атаками (см. пример 1). Изначально все юниты обладают максимальным запасом здоровья.

Вася точно помнит, что он выиграл в этой битве, то есть уничтожил армию противника. Однако, он не помнит начальное состояние армий. Вася знает список юнитов, которые были на поле боя, но не помнит, какие именно юниты были в его армии, какие — в армии противника и количество юнитов в отрядах. Кроме того, у Васи остался лог битвы, описывающий атаки отрядов. Для каждой атаки известны название атакующих юнитов, название атакуемых юнитов и величина нанесенного урона. Ваша задача — помочь Васе восстановить начальное положение армий по данным записям.

Формат входного файла

В первой строке задано единственное число N ($2 \leq N \leq 10$) — количество различных юнитов, находившихся на поле битвы. В каждой из следующих N строк находится описание соответствующего юнита. Сначала идет имя юнита $name_i$ — непустая строка длины не более 20, состоящая из строчных и прописных букв латинского алфавита. Далее в строке находятся 3 целых числа: количество единиц здоровья юнита H_i ($1 \leq H_i \leq 300$), левая и правая границы урона юнита L_i, R_i ($1 \leq L_i \leq R_i \leq 50$). Имена всех юнитов различны.

В следующей строке находится единственное число M ($1 \leq M \leq 1000$) — количество записей в логге. Каждая из следующих M строк представляет собой одну запись лога в виде:

$name_i$ deal D damage to $name_j$

где $name_i$ — имя атакующих юнитов, $name_j$ — имя атакуемых юнитов, D — нанесенный урон ($1 \leq D \leq 50000$). Гарантируется, что в битве не было двух отрядов с одинаковыми юнитами. При нанесении урона большего, чем суммарное количество здоровья у юнитов атакуемого отряда, будет указан именно нанесенный урон, а не полученный (см. пример 2). Записи в логге даны в хронологическом порядке (начиная с самой ранней).

Формат выходного файла

В первой строке нужно вывести количество отрядов в армии Васи. Далее необходимо вывести описание отрядов по одному в строке. Описание должно иметь вид:

$name_i$ K

где $name_i$ — название юнитов отряда, K — количество юнитов в отряде. Далее следует вывести описание армии противника Васи в том же формате. В каждой армии должно находиться не более 5 отрядов. В каждом отряде должно находиться не менее 1 и не более 1000 юнитов.

Описание должно соответствовать логу, то есть должна существовать такая последовательность выбранных уронов юнитов, что лог битвы будет иметь указанный вид. Гарантируется, что хотя бы одно решение существует. Если решений несколько, то можно вывести любое из них.

Примеры

stdin	stdout
3 Vampires 30 5 7 Minotaurs 35 5 10 Liches 25 8 10 5 Vampires deal 7 damage to Minotaurs Minotaurs deal 9 damage to Vampires Vampires deal 7 damage to Minotaurs Liches deal 16 damage to Minotaurs Vampires deal 6 damage to Minotaurs	2 Liches 2 Vampires 1 1 Minotaurs 1
2 Titans 300 20 30 Sprites 2 1 2 2 Sprites deal 1 damage to Titans Titans deal 9000 damage to Sprites	1 Titans 300 1 Sprites 1

Примечание

По счастливой случайности, в тестах используются только характеристики юнитов, совпадающие с юнитами из известной игры Heroes of Might and Magic II.

Разбор задачи F. HOLM 2: The Great Battle

Построим граф, вершинами которого являются существа, а ребрами - “атаки”, то есть если один отряд существ атаковал другой, то соединим соответствующие вершины ребром (неориентированным). Граф будет двудольным. Нам необходимо определить для каждой компоненты связности этого графа, какая из долей относится к армии Васи (т.е. к победившей армии). Для этого возьмем последнюю атаку, соответствующую компоненте связности. Атакующее существо этой атаки точно выжило (поскольку его никто не атаковал в дальнейшем), значит, относится к победившей армии.

Соответственно, и вся доля относится к армии Васи, а другая доля - к армии врага Васи.

Теперь необходимо определить количество существ в отрядах. У нас имеется 3 ограничения:

1. При атаке количество существ ограничивается нанесенным уроном.
2. Существо должно пережить все атаки (кроме, возможно, последней, если оно не атаковало после этого)
3. Существа проигравшей армии должны погибнуть.

Решим задачу для каждого отряда отдельно. Будем считать, что изначальное количество юнитов в нем неизвестно и находится в интервале от L до R (изначально, $L = 1$, $R = 1000$). Далее будем рассматривать записи в логах, связанные с данным отрядом, в хронологическом порядке. Будем поддерживать суммарный урон DR , который получил отряд. Тогда каждый раз, когда отряд атакует, мы находим границы $[L', R']$ количества юнитов в отряде на момент атаки (для этого нужно разделить урон отряда на минимальный/максимальный урон, наносимый юнитами отряда). Кроме того, уже $X = DR/H_i$ юнитов отряда были уничтожены до атаки. Тогда мы получили новые границы $L1 = L' + X$, $R1 = R' + X$ и полученный отрезок $[L1, R1]$ нужно пересечь с $[L, R]$.

Отдельно для каждого отряда нужно проверить ограничения 2 и 3, возможно, сузив при этом интервал $[L, R]$. Тогда любое количество существ из интервала $[L, R]$ будет удовлетворять условию задачи.

Задача G. Failout Few Vegas: Slow Save

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вася играет в Failout Few Vegas. В данной задаче суть игры и достижения Васи нам не важны, все, что нам важно знать — то что Вася решил сохранить игру. В диалоге сохранения ему было предложено задать имя, под которым текущая игра будет сохранена. Вася знает, под каким именем S он хочет сохранить игру. Однако, Вася очень ленив... Он решил набрать имя одним пальцем. Кроме того, он будет перемещать этот палец по клавиатуре слева-направо, нажимая в каждом столбце ровно одну клавишу (добавляя тем самым одну букву к текущей строке, представляющей имя). Клавиатура у Васи имеет 3 ряда по N символов в каждом. Таким образом, полученное имя сохраненной игры будет состоять ровно из N букв. Кроме того, каждый раз Вася перемещает палец либо на тот же ряд, где он уже

находится, либо на соседний с ним (первое нажатие может быть сделано в любом ряду). Разумеется, при этом не всегда можно получить желаемое название игры S . Вася определил "похожесть" имени M на строку S следующим образом: рассмотрим все непустые префиксы строки S , для каждого префикса найдем количество его вхождений в M и просуммируем эти числа. Полученное число и будет являться "похожестью" имени M на строку S . Помогите Васе определить, какую максимальную похожесть имени он может получить, действуя описанным выше алгоритмом.

Формат входного файла

В первой строке находится желаемое имя игры — непустая строка S , состоящая из строчных букв латинского алфавита. Длина строки S не превосходит 2000 символов. В следующих трех строках описана клавиатура Васи — по одному ряду в строке. Каждый ряд представляет собой непустую строку, состоящую из строчных букв латинского алфавита. Длины рядов равны между собой и не превосходят 2000. Клавиатура Васи уникальна и может содержать одинаковые символы.

Формат выходного файла

Вывести единственное число — наибольшую похожесть имени на строку S , которую может получить Вася.

Примеры

stdin	stdout
a aa aa aa	2
aa abc cab bca	5

Разбор задачи G. Fallout Few Vegas: Slow Save

В задаче нам пригодятся значения префикс-функции для строки S , поэтому их надо заранее найти — P_i . Будем решать задачу методом динамического программирования. Пусть $d[x][y][pref]$ — наибольшая похожесть, которую мы можем получить, начиная с позиции x в нашей строке, если мы уже поставили x букв в строке находимся в ряду y клавиатуры и наибольший суффикс уже построенной нами строки на первых $(x - 1)$ рядах

клавиатуры, совпадающий с префиксом строки S имеет длину $pref$. Переберем, в каком ряду i мы нажмем следующую клавишу. Для каждого возможного нажатия обновим длину наибольшего префикса $pref_i$. Теперь посчитаем, сколько префиксов строки S могут заканчиваться в позиции x нашей строки (пусть это число K_i). Понятно, что префиксы длины большей $pref_i$ не могут закончиться в x (по определению параметра $pref_i$). Если $pref_i$ равно 0, то, очевидно, $K_i = 0$. Иначе, в позиции x могут заканчиваться префиксы S длинами $pref_i, P[pref_i], P[P[pref_i]]$ и так далее, пока на очередной итерации не получим 0. Значения PP_i , равные длине последовательности $P[i], P[P[i]], \dots$ можно посчитать при вычислении префикс-функции строки S за $O(N^2)$, где N – длина строки S . Найдя K_i и $pref_i$ для каждого номера ряда i , мы сможем найти значение $d[x][y][pref]$, а именно: $d[x][y][pref] = \max(d[x+1][i][pref_i] + K_i)$. Соответственно, $d[n][i][pref] = 0$, а ответ задачи – $d[0][1][0]$.

Задача Н. Carmarandom TDC2013: New Trace

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вася играет в Carmarandom Total Destruction Contest 2013. В этой игре нужно управлять автомобилем, давить зомби, уничтожать противников, то есть играть в нее очень весело. Было весело. Вася прошел игру и новых целей больше нет. Однако, игра очень понравилась Васе, он не хочет ее забрасывать. Вася решил создать новую трассу для игры с опасными участками и резкими поворотами. Создав трассу, Вася сможет еще некоторое время наслаждаться любимой игрой, соревнуясь на этой трассе.

Вася хочет, чтобы трасса состояла из N прямых участков, соединенных между собой поворотами. Трасса не должна быть зациклена, то есть старт и финиш находятся в разных местах. Для усложнения движения по трассе Вася хочет сделать так, чтобы каждый прямой участок пересекался ровно с одним другим участком этой трассы по внутренней точке (то есть, не в повороте). Помогите Васе — по заданному числу N Вам необходимо предложить вариант трассы, состоящей из N участков и удовлетворяющей описанным условиям.

Формат входного файла

Входной файл содержит единственное число N — количество прямых участков на трассе ($10 \leq N \leq 100$).

Формат выходного файла

Если можно создать трассу требуемым образом, то в первой строке нужно вывести 'Yes' (без кавычек). Далее следует вывести $N + 1$ пару целых чисел, по одной паре в строке. Первая пара соответствует координатам старта трассы, следующие $N - 1$ пара соответствуют координатам поворотов (в порядке их следования по трассе), последняя пара соответствует координатам финиша. Старт и финиш должны различаться. Каждый прямой участок трассы должен пересекать ровно один другой участок по внутренней точке. Расстояния от точки пересечения до концов пересекающихся участков должны быть не менее 0.5. Повороты должны быть совершены на углы большие 0 и меньше 180 градусов, то есть никакие три подряд идущие выведенные точки не должны лежать на одной прямой. Координаты должны быть целыми и не превосходить 10000 по абсолютному значению.

Если искомой трассы не существует, то необходимо вывести 'No' (без кавычек).

Примеры

stdin	stdout
11	No

Разбор задачи H. Carmarandom TDC2013: New Trace

Нам необходимо построить N -звенную незамкнутую ломаную, которая пересекает каждое свое звено ровно 1 раз. Звенья ломаной разбиваются на пары пересекающихся между собой, значит, N должно быть четно. Построим пример для $N = 6$:

```
-2 -1
-1  2
 3  3
 0  0
-3  3
 1  2
 2 -1
```

Продолжая ломаную в обе стороны, легко получаем ответ для любого четного N .

Задача I. X-Dom: Railway

Вход:	stdin
Выход:	stdout
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Вася играет в X-Dom: The Distance Unknown. В этой игре Васе предстоит спасти Землю от инопланетного вторжения. Для этого он объединил в союз все страны от Германии до Китая. Для быстрого перемещения войск между ними была построена прямая железная дорога. Однако, пришельцы решили разрушить транспортную систему Васи и отправили на Землю N энергетических станций. Далее был запущен механизм уничтожения железной дороги. Происходило это следующим образом: для каждой пары станций в середине отрезка, соединяющего их, создавался мощный заряд. Далее из каждого такого заряда выпускалось два разрушающих луча, лучи летели по прямой, ортогональной, соответствующему двум станциям, отрезку, и были направлены в противоположные стороны. Если луч пересекал железную дорогу, то он повреждал ее в точке пересечения. Вам необходимо оценить масштабы разрушений, а именно найти расстояние между двумя наиболее удаленными поврежденными участками дороги.

Формат входного файла

В первой строке дано целое число N ($2 \leq N \leq 75000$) — количество инопланетных станций. В каждой из следующих N строк даны два целых числа X_i и Y_i , по модулю не превосходящие 100000, — абсцисса и ордината положения соответствующей станции. Все станции находятся в различных точках и не лежат на оси ОХ. Железная дорога Васи идет вдоль прямой $Y = 0$ и является бесконечной в обоих направлениях. Гарантируется, что хотя бы один из полученных разрушающих лучей пересекает ось ОХ. Гарантируется, что ни один из разрушающих лучей не совпадает с осью ОХ.

Формат выходного файла

Вывести единственное вещественное число с абсолютной или относительной погрешностью не более 10^{-6} — искомое расстояние между двумя наиболее удаленными поврежденными участками дороги.

Примеры

stdin	stdout
3 0 1 -2 5 2 5	14.000000

Примечание

Примечание 1: В примере заряды создаются в точках $(-1, 3)$, $(0, 5)$, $(1, 3)$, а лучи повреждают дорогу в точках $(-7, 0)$, $(0, 0)$ и $(7, 0)$ соответственно.

Примечание 2: Входные данные могут иметь размер больше 1МБ, не рекомендуется использовать объект *cin* для чтения входных данных. Чтение большого объема данных с использованием *cin* требует достаточно много процессорного времени, что может привести к вердикту «*Time Limit Exceeded*».

Разбор задачи I. X-Dom: Railway

Прямая, содержащая серединный перпендикуляр к отрезку, делит плоскость на 2 части. В первой части находятся точки, которые ближе к первому концу отрезка, во второй - ко второму. Для каждой точки P на прямой O_X мы можем определить порядок O_P исходных точек относительно нее следующим образом: точки будут упорядочены в порядке увеличения расстояния до P . Будем двигать точку P по оси O_X . Тогда при переходе P через точку пересечения O_X и одного из серединных перпендикуляров порядок O_P изменится. Порядок O_P перестанет меняться при движении точки P после того, как P перейдет через последнюю точку пересечения одного из серединных перпендикуляров с прямой O_X . Зафиксируем этот порядок O_P (соответствующий точке $P = (+\infty, 0)$). Тогда бинарным поиском можно найти самую левую точку P' на оси O_X , для которой $O_{P'}$ будет совпадать с O_P . Найденная точка будет совпадать с самой правой точкой пересечения серединного перпендикуляра и O_X . Проверку совпадения порядка можно сделать за $O(N)$, бинарный поиск требует $O(\log(\text{Max}X))$ операций. Аналогично мы можем найти самую левую точку, ответом будет являться расстояние между самой правой и самой левой точками.

Задача J. MindCraft: Heliport

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 3 с
 Ограничение по памяти: 256 Мб

Вася играет в MindCraft. Вася выбрал себе поле размером $N \times M$ клеток, на каждой клетке которого может стоять несколько единичных кубиков. Количество кубиков, стоящих на определенной клетке, назовем высотой этой клетки. За одну секунду Вася может поставить на любую клетку один кубик или убрать с клетки один кубик (если на ней было положительное число кубиков), т.е. изменить высоту клетки на 1. Вася хочет, чтобы на его поле появилась вертолетная площадка размером $H \times W$ клеток. Для этого Васе нужно, чтобы на его поле был прямоугольник размером $H \times W$, все клетки которого имели бы одинаковую высоту. В этом случае площадка получится ровной и Вася будет крайне доволен. Ваша задача — определить, за какое наименьшее время Вася сможет построить такую площадку.

Формат входного файла

В первой строке находятся два целых числа N и M ($1 \leq N, M \leq 500$) — размеры поля, на котором играет Вася. В следующей строке находятся два целых числа H и W ($1 \leq H \leq 20$; $1 \leq W \leq 200$; $H \leq N, W \leq M$) — размеры вертолетной площадки. Далее следуют N строк по M чисел в каждой — начальные высоты клеток. Высоты клеток неотрицательные и не превосходят 100000.

Формат выходного файла

Вывести единственное число — минимальное время, за которое Вася сможет подготовить вертолетную площадку.

Примеры

<code>stdin</code>	<code>stdout</code>
2 3 1 2 1 2 3 4 5 6	1
2 3 1 2 1 6 3 4 2 5	2

Примечание

В первом примере площадку можно строить в любом месте — всегда нужно будет добавить (или убрать) один кубик.

Во втором примере Вася может убрать из первой клетки второго ряда два кубика, получив тем самым площадку размером 1×2 высоты 2.

Входные данные могут иметь размер больше 1МБ, не рекомендуется использовать объект *cin* для чтения входных данных. Чтение большого объема данных с использованием *cin* требует достаточно много процессорного времени, что может привести к вердикту «*Time Limit Exceeded*».

Разбор задачи J. MindCraft: Heliport

Пусть мы зафиксировали позицию прямоугольника, соответствующего искомой площадке. Найдем, к какой высоте необходимо приводить все клетки прямоугольника. Пусть X - множество высот клеток. Тогда легко проверить, что наименьшее число действий будет выполнено, если мы приведем высоты к числу Y , являющемуся медианой множества X . Нам важно, за сколько времени мы можем это сделать. Для быстрого нахождения этого значения нужно знать следующие числа: K_1 - количество чисел множества X , больших Y , S_1 - сумма этих чисел, K_2 - количество чисел множества X , меньших Y , S_2 - сумма этих чисел.

Тогда время T , необходимое для выравнивания площадки, будет равно $T = S_1 - K_1 \cdot Y + K_2 \cdot Y - S_2(*)$.

Будем передвигать положение площадки по исходному полю, поддерживая значения K_1 , S_1 , K_2 , S_2 . Для этого все числа, соответствующие высотам площадки, разобьем на два мультисета MS_1 и MS_2 , таких что их размеры отличаются не более, чем на 1 ($Size(MS_2) \leq Size(MS_1) \leq Size(MS_2) + 1$) и каждое число MS_1 больше каждого числа MS_2 . Тогда медианой множества X будет наименьшее число из MS_1 . K_1 будет равно размеру MS_1 , а S_1 - сумме чисел MS_1 . Аналогичные утверждения выполняются и для MS_2 . Изменяя мультисеты MS_1 и MS_2 нужно пересчитывать суммы S_1 и S_2 . Поместим площадку в левый-верхний угол и будем двигать вправо по одной клетке. На каждом шаге будем пытаться найти более оптимальный ответ на задачу в соответствии с формулой (*). При движении вправо мы будем удалять H клеток из нашего множества X и добавлять H новых клеток (соответственно, меняются и мультисеты MS_1 , MS_2). Одно число можно удалить или добавить из множества X за $O(\log(H * W))$. Значит, один шаг вправо можно выполнить за $O(H * \log(H * W))$. Дойдя до правой границы нужно сделать шаг вниз аналогичным образом за $O(W * \log(H * W))$. Да-

лее будем двигаться влево до границы и т.д. Таким образом, мы обойдем все возможные позиции площадки сделав порядка $O(N \cdot M)$ шагов влево/вправо и $O(N)$ шагов вниз. Общая сложность алгоритма будет равна $O(N \cdot M \cdot H \cdot \log(H \cdot W) + N \cdot W \cdot \log(H \cdot W)) = O(N \cdot M \cdot H \cdot \log(H \cdot W))$.

Задача К. Double Life: Amplifiers

Вход:	stdin
Выход:	stdout
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Вася играет в Double Life. Ему нужно уничтожить очень сильного монстра — легкраба. У Васи есть мощное оружие — вау-пушка. Кроме того, в комнате, в которой находятся Вася и монстр, действует N полей от находящихся неподалеку усилителей. Каждое поле характеризуется коэффициентом — целым числом A_i , показывающим, насколько усилитель изменяет мощность выстрела вау-пушки. Если значение A_i положительно, то нанесенный урон увеличится в A_i раз. При A_i равном 0, действие поля приводит к тому, что мощность выстрела будет равна 0 и цель не получит урон. Самое опасное — если A_i отрицательно. В этом случае мощность выстрела увеличится в $|A_i|$ раз, однако, направление выстрела изменится на противоположное. Когда Вася производит выстрел, то к нему (выстрелу) последовательно применяется действие всех полей (легко заметить, что при данных условиях порядок применения полей не важен). Если после применения всех полей направление заряда осталось прежним, то он летит в легкраба. Иначе, заряд летит в противоположном направлении — то есть, в Васю.

Вася не обязан атаковать немедленно. У него есть K секунд, прежде чем монстр уничтожит его. Вася может использовать свободное время для изменения коэффициентов усилителей. За одну секунду он может изменить одно значение A_i на 1 (Вася может как увеличить A_i , так и уменьшить). Разумеется, Вася хочет нанести наибольший урон легкрабу. Если же в любом случае заряд полетит в самого Васю, то необходимо сделать полученный урон как можно меньшим. Помогите Васе определить, какой же урон после изменения коэффициентов усилителей может быть нанесен в наилучшем случае.

Формат входного файла

В первой строке находится число N — количество усиливающих полей ($1 \leq N \leq 50000$). Во второй строке через пробел перечислены коэффици-

енты усиления A_i ($-100000 \leq A_i \leq 100000$). В третьей строке находится число K — количество свободного времени у Васи ($0 \leq K \leq 10^9$).

Формат выходного файла

Выведите единственное число — наилучшую итоговую мощность выстрела, которую может достичь Вася. Если заряд будет направлен в Васю, то перед мощностью заряда нужно поставить знак '-' (см. пример 2). Изначальная мощность заряда (без усиления) равна 1.

Примеры

stdin	stdout
3 1 0 1 2	2
3 10 10 -5 3	-200
3 0 0 0 2	0

Разбор задачи К. Double Life: Amplifiers

По условию задачи, нам дано N чисел и мы должны изменить их в сумме на K , так чтобы произведение было максимальным. Нам не важны знаки чисел, важна только четность количества отрицательных чисел. При этом нам не важно, какие именно числа отрицательные. Можно считать, что у нас есть либо 0 отрицательных чисел, либо 1 отрицательное число, причем отрицательное число минимально по модулю (в частности, может быть -0). Верно следующее утверждение: пока $K > 0$ мы должны увеличить наименьшее число из множества на 1 (соответственно, уменьшить K на 1). Чтобы доказать это утверждение рассмотрим 4 случая.

1. Все числа положительны. Для формального доказательства этого очевидного утверждения воспользуемся неравенством Караматы (http://en.wikipedia.org/wiki/Karamata%27s_inequality). А именно, возьмем логарифм от произведения наших чисел. Поскольку логарифм — монотонная функция, то нам нужно максимизировать сумму логарифмов наших чисел. Заметим, что логарифм — выпуклая функция. Значит, если набор чисел $A = a_1, \dots, a_n$ мажорирует набор чисел $B = b_1, \dots, b_n$, то $\log(a_1) + \dots + \log(a_n) \leq \log(b_1) + \dots + \log(b_n)$. Легко проверить, что при

данном алгоритме для любого K полученный набор чисел будет мажорироваться любым другим набором, который можно получить, изменив исходные числа не более, чем на K .

2. Все числа неотрицательны. Пусть у нас есть X чисел равных 0. Если $K < X$, то мы в любом случае получим ответ 0. Если $K \geq X$, то для получения положительного ответа мы должны каждое из X чисел увеличить хотя бы на 1 (K уменьшится на X). Тем самым, мы получим ситуацию из пункта 1.

3. Одно число отрицательно, по модулю равно M , $M \leq K$. Очевидно, что для того, чтобы получить положительный ответ, мы должны изменить знак одного числа. Наиболее выгодно изменить знак наименьшего по модулю числа, то есть отрицательного числа. После мы получим ситуацию из пункта 2.

4. Одно число отрицательно, по модулю равно M , $M > K$. В этом случае мы получаем отрицательный ответ, то есть нам нужно минимизировать модуль произведения. Действуя описанным алгоритмом мы получим набор, который будет мажорировать любой другой набор, измененный на K (поскольку мы уменьшили на K минимальное число). Значит, воспользовавшись неравенством из пункта 1, мы получим наименьшее возможное произведение (по модулю).

Теперь нужно быстро выполнить описанный алгоритм. Для этого сначала отсортируем исходные числа. Будем идти от маленьких чисел к большим, поддерживая количество текущих минимальных элементов. На каждом шаге мы пытаемся приравнять все минимальные элементы к следующему по величине элементу. Если K достаточно велико, чтобы выполнить это, то мы увеличиваем количество минимальных элементов и повторяем алгоритм, иначе выходим. Итого, за время $O(N)$ мы нашли оптимальные значения. Далее нужно перемножить эти числа. Для этого введем функцию f , которая перемножает числа на интервале $[L, R)$ и возвращает результат. Она будет работать следующим образом: пусть $C = (L + R)/2$, далее $X_1 = f(L, C)$, $X_2 = f(C, R)$. Найдем произведение $X = (X_1 * X_2)$ при помощи быстрого преобразования Фурье, и вернем X . Ответом на задачу будет $f(0, N)$. Легко проверить, что сложность работы алгоритма перемножения есть $O(N * \log^2(N))$.

День четвертый (19.02.2013 г.) Контеcт Неспирного Виталия Николаевича

Об авторе...

Неспирный Виталий Николаевич, кандидат физико-математических наук (специальность 01.02.01 “Теоретическая механика”), докторант Института прикладной математики и механики НАН Украины, старший научный сотрудник отдела технической механики, ученый секретарь специализированного ученого совета по защите диссертаций на соискание ученого звания доктора наук (Д 11.193.01) по специальности 01.02.01, доцент кафедры теории упругости и вычислительной математики Донецкого национального университета, заместитель председателя жюри II-III этапов Всеукраинской олимпиады школьников по информатике в Донецкой области, член жюри Всеукраинской олимпиады школьников по информатике, председатель жюри по информатике Всеукраинской комплексной олимпиады по математике, физике и информатике “Турнир чемпионов”, координатор Донецкого сектора Открытого кубка по программированию, тренер команд математического факультета Донецкого национального университета.



Основные достижения:

- 1-е место на Всеукраинской студенческой олимпиаде по информатике (Николаев, 2000);
- 1-е место на Всеукраинской АСМ-олимпиаде по программированию в составе команды математического факультета ДонНУ (Винница, 2001);
- подготовил 3 призеров международных школьных олимпиад;
- совместно с А.И.Парамоновым подготовил команду DonNU United, занявшую 7-е (2008), 4-е (2009) и 7-е место (2010) в ACM SEERC, а в 2011 году 8-е место (серебряные медали) в финале ACM ICPC.

- тренер команды SCH_Donetsk Erudit, занявшей 1-е место в школьном зачете XII Открытого Кубка по программированию им. Е.В.Панкратьева (2013)

Теоретический материал. Линейное программирование и матричные игры

Задача линейного программирования

Общая задача линейного программирования заключается в отыскании вектора (x_1, x_2, \dots, x_n) , минимизирующего (или максимизирующего) линейную форму

$$c_1x_1 + \dots c_nx_n, \quad (1)$$

Переменные которой подчинены ограничениям вида

$$a_{i1}x_1 + a_{i2}x_2 + \dots a_{in}x_n ? b_i, \quad i = \overline{1, m} \quad (2)$$

где вместо знака ? может стоять один из знаков $=$, \leq либо \geq , а a_{ij} , b_i , c_j – заданные постоянные.

Функцию (1) принято называть целевой функцией, любой вектор, удовлетворяющий условиям (2), планом задачи, а оптимальным планом – план, оптимизирующий целевую функцию (1).

Теорема 1. Множество всех планов задачи линейного программирования является выпуклым.

Это означает, что если две точки принадлежат множеству планов, то и весь отрезок соединяющий эти точки также принадлежит этому множеству. Точка называется крайней, если она не является внутренней точкой никакого отрезка с концами, принадлежащими множеству планов. Легко проверить, что любая крайняя точка должна быть граничной, но не всякая граничная точка является крайней.

Теорема 2. Целевая функция достигает своего минимума или максимума в выпуклой ограниченной области K по меньшей мере в одной из крайних точек этой области. Если оптимальное значение принимается более чем в одной крайней точке, то функция достигает этого же значения в любой точке, являющейся выпуклой комбинацией этих крайних точек.

Таким образом, достаточно перебрать все крайние точки множества планов и выбрать из них оптимальное. Каждую крайнюю точку можно найти следующим образом – выбрать n линейно независимых уравнений (2), заменив в них возможно знак неравенства на равенство, решить их. Если полученная точка удовлетворяет остальным ограничениям, то она является

планом задачи и при этом крайней точкой множества планов. Следовательно, придется перебрать не более $\binom{m}{n}$ точек.

Однако можно перебирать крайние точки более системно, выбирая следующую точку таким образом, чтобы значение целевой функции в ней было лучше (или по крайней мере не хуже) чем в предыдущей. Для этого задачу следует привести к каноническому виду.

Канонический вид задачи линейного программирования

Канонической формой задачи линейного программирования называется такая форма, при которой требуется минимизировать функцию (1), все ограничения (2) являются равенствами, причем все b_i неотрицательны. И кроме того, все переменные должны быть неотрицательными, т.е. добавляются ограничения $x_j \geq 0$.

Любую задачу линейного программирования можно свести к каноническому виду. Разберем преобразования, которые необходимо для этого выполнить.

1. Задача на максимум. Если требуется найти максимум целевой функции, то заменим все c_j на $-c_j$. План, который максимизировал старую целевую функцию, для новой функции станет планом, доставляющим минимум.

2. Ограничения с отрицательной правой частью. Если некоторое ограничение имеет отрицательное значение b_i , то достаточно опять же домножить все его коэффициенты на -1 (и заменить знак неравенства на противоположный), если ограничение не было равенством.

3. Отрицательные переменные. Если на некоторую переменную x_j в исходной задаче наложено ограничение вида $x_j \geq b_j$, то за счет замены $x_j = x'_j + b_j$, можно это ограничение превратится в $x'_j \geq 0$. Если имелось ограничение вида $x_j \leq b_j$, то замена $x_j = -x'_j + b_j$ превратит его в $x'_j \geq 0$. Ну и наконец, если на переменную x_j нет явного ограничения, можно сделать замену $x_j = x'_j - x''_j$, и на введенные таким образом переменные наложить ограничения $x'_j \geq 0, x''_j \geq 0$.

4. Превращение равенств в неравенства. Пусть некоторое ограничение задается равенством вида:

$$a_{i1}x_1 + a_{i2}x_2 + \dots a_{in}x_n \leq b_i.$$

Тогда введем новую переменную x_{n+1} и перепишем это неравенство в виде:

$$a_{i1}x_1 + a_{i2}x_2 + \dots a_{in}x_n + x_{n+1} = b_i,$$

и добавим требуемое канонической формой задачи ограничение $x_{n+1} \geq 0$. Если же исходное неравенство было \geq , то в это ограничение переменная

x_{n+1} войдет с коэффициентом -1 . Для каждого неравенства нужно вводить свою отдельную переменную. Все введенные таким образом дополнительные переменные войдут в целевую функцию с нулевым коэффициентом.

Для задачи линейного программирования в канонической форме справедливы следующие утверждения.

Теорема 3. Если известно, что система векторов P_1, P_2, \dots, P_k линейно-независима, где P_i векторы-столбцы системы уравнений (2), и выполняется

$$x_1 P_1 + \dots + x_k P_k = P_0,$$

то точка $X = (x_1, \dots, x_k)$ – является крайней точкой множества планов.

Теорема 4. Если $x = (x_1, \dots, x_n)$ – крайняя точка множества планов, то векторы P_i , соответствующие ненулевым значениям x_i , образуют линейно-независимую систему.

Отсюда следует, что крайняя точка имеет не более чем m положительных компонент.

Симплекс-метод

Предположим, что у нас в каждом уравнении i имеется своя переменная $x_{g[i]}$, входящая в него с коэффициентом 1, а во все остальные уравнения не входит. Тогда векторы $P_{g[i]}$ образуют базис пространства R^n , и разумеется являются линейно-независимыми. Возьмем опорный план

$$x_j = \begin{cases} b_i & \text{если } i \text{ такое, что } g[i] = j, \\ 0 & \text{если } j \neq g[i] \text{ при всех } i. \end{cases} \quad (3)$$

Согласно теореме 3 он является крайней точкой множества планов. Все остальные векторы можно считать записанными в базисе $\{P_{g[i]}\}$.

Вычислим значения $z_j = \sum_{1 \leq i \leq m} c_{g[i]} a_{ij}$.

Теорема 5. Если для всех j выполняется условие $z_j \leq c_j$, то план (3) будет оптимальным.

Теорема 6. Если для некоторого j выполняется условие $z_j > c_j$, то план (3) можно улучшить. Для этого следует вычислить значение $\theta = \min_{i: a_{ij} > 0} b_i / a_{ij}$. Если соответствующий минимум не определен, то есть при всех i коэффициенты a_{ij} неположительны, то целевая функция может быть сделана сколь угодно малой. В противном случае, можно заменить в базисе вектор $P_{g[i]}$ на вектор P_j (разделив j уравнение на коэффициент a_{ij}), и затем вычитая из всех остальных уравнений j -ое уравнение, умноженное на коэффициент стоящий при переменной x_j в соответствующем уравнении. На опорном плане, определяющемся формулой (3) с учетом нового базиса, целевая функция будет иметь значение на $\theta(z_j - c_j)$ меньше, чем на старом плане.

Если существует несколько j , для которых $z_j > c_j$, то выбор j можно осуществлять произвольным образом. Как правило выбирают такое j , при котором величина $z_j - c_j$ максимальна. На самом деле, разумнее выбирать столбец j по максимуму величины $\theta_j(z_j - c_j)$, поскольку в этом случае мы получим лучший из следующих возможных опорных планов. Однако разумеется это не гарантирует, что такой выбор обеспечит более быструю сходимость к оптимальному плану.

Теорема 7. Начав с некоторого опорного плана, и действуя так, как описано в теореме 6, мы за конечное число шагов придем к оптимальному плану.

Число шагов очевидно будет ограничено значением $\binom{n}{m}$. И существует последовательность задач, для которых эта порядок этой оценки асимптотически достигается. Поэтому в худшем случае симплекс-метод работает за экспоненциальное время. Однако есть оценка, что число итераций симплекс-метода является величиной порядка $2m + n$. Это означает, что для решения задачи линейного программирования этим методом потребуется в среднем $O(mn(m + n))$ операций.

Искусственный базис

Если матрица системы (2) не содержит единичной подматрицы размера m , то недостающие векторы вводят искусственно, добавляя в соответствующее уравнение переменную и вводя ее в целевую функцию с коэффициентом w , которое мы предполагаем достаточно большим числом. Решая такую задачу, мы должны будем вычислять значения z_j в виде $z_j^{(1)}w + z_j^{(2)}$. Сравнить такие значения мы должны будем лексикографически, поскольку считаем w очень большим числом.

Если в базисе, соответствующем оптимальному плану такой задачи, останутся искусственные векторы, то можно утверждать, что исходная задача имела пустое множество планов. В противном случае оптимальный план задачи с искусственными переменными даст нам решение исходной задачи.

Матричные игры

Рассмотрим игру для двух игроков, которая имеет нулевую сумму. Это означает, что выигрыш первого игрока равен проигрышу второго. Каждый игрок может иметь некоторые возможные стратегии. Стратегией называется некоторый план на игру, настолько исчерпывающий, что он не может быть нарушен ни при каких условиях, ни при каких действиях противника или природы, но может учитывать их. Разумеется исход игры будет полностью определяться выбором своих стратегий каждым из игроков. Мы

будем рассматривать игры, в которых каждый игрок имеет конечное число стратегий. Допустим у первого игрока может быть m возможных вариантов выбора ведения игры, а у второго – n . Такую игру будем называть игрой $m \times n$.

Платежной матрицей или просто матрицей игры называется матрица, элемент a_{ij} которой равен выигрышу первого игрока (а значит и проигрышу второго) при выборе первым игроком своей i -ой стратегии, а вторым игроков j -ой стратегии. Считается, что игроки обладают полной информацией об игре, то есть знают ее правила, знают все возможные стратегии друг друга и платежную матрицу. Но разумеется о том, какой выбор стратегии сделает их соперник они заранее знать не могут.

Решение матричных игр

Решить матричную игру – означает дать определенное правило выбора стратегии для каждого игрока, таким образом, чтобы его наименьший выигрыш был как можно больше при любом варианте игры соперника. Это означает, что все рассуждения проводятся в предположении, что наш соперник играет оптимально. Разумеется, если приписать сопернику определенную степень невежества или глупости, правило выбора стратегии может измениться существенно. Но теория игр не приписывает сопернику этих качеств и потому является консервативной (осторожной) теорией.

Наиболее просто решаются игры, в матрице которых есть седловая точка. Рассмотрим, например, игру с матрицей

	1	2	3	4
1	7	2	5	1
2	2	2	3	4
3	5	3	4	4
4	3	2	1	6

Максимальный выигрыш 7 первому игроку могла бы принести стратегия 1, но совершенно ясно, что второй игрок вряд ли будет выбирать стратегию 1 которая приводит к такому результату. В худшем случае, игрок 1 рискует получить выигрыш в размере всего лишь 1, если будет пользоваться стратегией 1. При выборе стратегии 2 таким значением будет значение 2, при стратегии 3 – значение 3, при стратегии 4 – значение 1. Таким образом, вариант выбора стратегии 3 обеспечивает первому игроку выигрыш не меньше 3. Рассмотрим теперь игру с точки зрения второго игрока. При выборе стратегии 1 он рискует проиграть 7, при выборе стратегии 2 – проигрыш будет не больше 3, при стратегии 3 – не больше 5, при стратегии 4 – не больше 6. Таким образом, оптимальной стратегией для второго игрока является стратегия 2, которой он может обеспечить себе проигрыш не

больше 3. Итак, мы имеем, что стратегия 3 оптимальна для первого игрока, а стратегия 2 – для второго. При выборе этих стратегий первый игрок получает выигрыш 3 (эта величина называется ценой игры), и если кто-то из игроков отклонится от своей оптимальной стратегии, то рискует выиграть меньшую сумму (или потерять больше), чем при своей оптимальной стратегии.

Обобщая это на произвольную игру, может сделать следующее утверждение: игрок 1 может играть так, чтобы обеспечить себе выигрыш не меньше, чем $\max_i \min_j a_{ij}$, второй игрок может играть так, чтобы гарантировать, что его проигрыш будет не выше величины $\min_j \max_i a_{ij}$. Если эти значения равны, в матрице есть седловая точка и игра имеет решение в так называемых чистых стратегиях. В такой игре игроки могут даже не скрывать друг от друга выбор своей стратегии, поскольку такая информация не может принести пользы сопернику (разумеется если это оптимальная стратегия).

Совсем иначе дело обстоит, когда $\max_i \min_j a_{ij} < \min_j \max_i a_{ij}$. В этом случае мы можем утверждать, что цена игры где-то между этими значениями. И игроку раскрывать свой выбор противнику заранее нельзя, поскольку тот может воспользоваться этим выбором для того, чтобы повысить свой выигрыш. В этом случае используют так называемые смешанные стратегии, когда выбор игрока определяется случайным образом из нескольких стратегий.

Смешанной стратегией первого игрока называется набор вероятностей (p_1, p_2, \dots, p_m) , таких что $p_i \geq 0$ и $\sum p_i = 1$. Здесь p_i – вероятность выбора игроком своей i -ой стратегии. Если первый игрок будет использовать свою смешанную стратегию p , а второй игрок – смешанную стратегию q , то в среднем первый игрок будет получать $V(p, q) = \sum a_{ij} p_i q_j$. Такое расширение множества стратегий приводит к тому, что в игре появляется седловая точка, о чем говорится в следующей теореме.

Теорема Дж. фон Неймана. Для любой матричной игры существуют смешанные стратегии p^* и q^* такие, что для любых смешанных стратегий p и q выполняется неравенство $V(p, q^*) \leq V(p^*, q^*) \leq V(p^*, q)$.

Для нахождения оптимальной смешанной стратегии можно воспользоваться следующей схемой. Пусть цена игры будет равна V . Это означает, что при любой чистой стратегии второго игрока, математическое ожидание выигрыша первого игрока, использующего свою оптимальную смешанную стратегию, будет не меньше V :

$$\begin{aligned} \sum a_{ij} p_i &\geq V, \quad j = \overline{1, n}, \\ \sum p_i &= 1, \\ p_i &\geq 0, \quad i = \overline{1, m}. \end{aligned} \tag{4}$$

Ясно, что если против каждой из чистых стратегий, первый игрок получает не меньше V , то эта оценка будет верна и против любой смешанной стратегии. Задача первого игрока заключается в том, чтобы максимизировать величину V . Это значит, что целевой функцией является $f(p, V) = V$, и нам нужно найти ее максимум при условиях (4). Эта задача является задачей линейного программирования и может быть решена с помощью симплекс-метода. Аналогично ищется оптимальная стратегия второго игрока.

Если есть гарантия, что цена игры положительна (а это будет, например, в случае, когда все элементы платежной матрицы положительны), то можно сделать замену переменных $p'_i = p_i/V$ и для нахождения оптимальной стратегии и цены игры решать задачу

$$\begin{aligned} \sum p'_i &= 1/V \rightarrow \min, \\ \sum a_{ij}p'_i &\geq 1, & j = \overline{1, n}, \\ p'_i &\geq 0, & i = \overline{1, m}. \end{aligned}$$

Следует отметить, что в случае, когда мы знаем как играет наш соперник (какую чистую или смешанную стратегию он применяет), оптимальную стратегию можно искать среди чистых стратегий. Отсюда следует в частности тот факт, что оптимальная стратегия дает нам один и тот же выигрыш как против соперника, использующего действительно свою оптимальную стратегию, так и против игрока, который играет даже неоптимальной чистой стратегией, которая однако входит с ненулевой вероятностью в оптимальную смешанную. Однако, если будет заранее известно какую стратегию применяет соперник и она не оптимальна, может быть подобрана стратегия, которая даст выигрыш больше, чем цена игры.

Задачи и разборы

Задача А. Неприводимые многочлены High

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Имеется некоторое простое число p . Возьмем $Z_p = \{0, 1, \dots, p-1\}$ – поле вычетов по модулю p . В этом поле операции умножения и сложения выполняются по модулю p . Теперь рассмотрим нормированные многочлены над этим полем вида

$$f(x) = x^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0,$$

где n – степень многочлена, x – переменная, $a_i \in Z_p$ – коэффициенты.

Так, например, многочлен $x^2 + x + 1$ в поле Z_2 неприводим. Нормированный многочлен называется неприводимым, если нельзя представить его в виде $f(x) = p(x) \cdot q(x)$, где $p(x)$, $q(x)$ – многочлены степени меньшей, чем $f(x)$. И это единственный неприводимый многочлен второй степени в поле Z_2 .

Ваша задача – вычислить количество неприводимых многочленов степени n в поле Z_p . Поскольку это количество может быть большим, то требуется получить лишь остаток от деления этого числа на m .

Ограничения

Числа p , n , m – целые, p – простое.

$1 \leq p, n, m \leq 10^9$.

Формат входного файла

Единственная строка входного файла содержит три числа p , n , m .

Формат выходного файла

Выведите в выходной файл количество неприводимых многочленов степени n в поле Z_p по модулю m .

Пример

stdin	stdout
2 2 10	1
3 4 100	18

Задача В. Неприводимые многочлены Junior

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Имеется некоторое простое число p . Возьмем $Z_p = \{0, 1, \dots, p-1\}$ – поле вычетов по модулю p . В этом поле операции умножения и сложения выполняются по модулю p . Теперь рассмотрим нормированные многочлены над этим полем вида

$$f(x) = x^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0,$$

где n – степень многочлена, x – переменная, $a_i \in Z_p$ – коэффициенты.

Так, например, многочлен $x^2 + x + 1$ в поле Z_2 неприводим. Нормированный многочлен называется неприводимым, если нельзя представить его в виде $f(x) = p(x) \cdot q(x)$, где $p(x)$, $q(x)$ - многочлены степени меньшей, чем $f(x)$. И это единственный неприводимый многочлен второй степени в поле Z_2 .

Ваша задача – вычислить количество неприводимых многочленов степени n в поле Z_p .

Ограничения

Числа p , n – целые, p – простое.

$1 \leq p, n \leq 10^9$; $1 \leq p^n < 10^{18}$.

Формат входного файла

Единственная строка входного файла содержит два числа p , n .

Формат выходного файла

Выведите в выходной файл количество неприводимых многочленов степени n в поле Z_p .

Пример

stdin	stdout
2 2	1
3 4	18

Разбор задачи В. Неприводимые многочлены

Рассмотрим некоторый нормированный многочлен $f(x)$. Если он не является неприводимым, то его можно разложить на два множителя $f(x) = p_1(x)p_2(x)$. Если какой-то из этих многочленов неприводим, то мы можем разбить его снова на множители и продолжать этот процесс до тех пор, пока не получим представление:

$$f(x) = p_1(x)p_2(x) \dots p_k(x).$$

Можно доказать для нормированных многочленов аналог основной теоремы арифметики о том, что такое представление будет единственным с точностью до порядка множителей.

Отсюда следует, что количество приводимых многочленов степени n (обозначим его $R(n)$) может быть вычислено по следующей формуле:

$$R(n) = \sum_{i_1+2i_2+\dots+(n-1)i_{n-1}=n} C(I(1), i_1)C(I(2), i_2) \dots C(I(n-1), i_{n-1}), \quad (*)$$

где $I(n) = p^n - R(n)$ – количество неприводимых многочленов степени n , а $C(n, k)$ – количество сочетаний с повторениями из n по k , которое вычисляется по формуле

$$C(n, k) = \binom{n+k-1}{k}.$$

В формуле (*) мы перебираем все возможные разбиения числа n на слагаемые, которые будут определять степени неприводимых делителей, и определяем количество вариантов выбора (возможно с повторениями) многочленов для каждой степени.

К сожалению количество разложений достаточно велико, и поэтому вычисления по формуле (*) потребуют достаточно много времени. Поэтому дадим нашей функции R еще один параметр. Пусть $R(n, k)$ – количество приводимых многочленов, которые имеют неприводимые делители лишь степени не выше k . Тогда $R(n, 0) = 0$ и $I(n) = p^n - R(n, n-1)$. Рекуррентную формулу легко получить, перебрав все возможные варианты количества неприводимых делителей многочлена степени k . В результате получим формулу:

$$R(n, k) = \sum_{i=0}^{\lceil n/k \rceil} C(I(k), i) R(n, k-i).$$

Для вычисления величины всех величин $R(i, k)$ для $1 \leq i \leq n$, $0 \leq k \leq i-1$ нам потребуется время порядка $O(n^2 \log n)$. Этого будет достаточно, чтобы решить задачу в варианте Junior, поскольку из ограничений p – простое и $p^n \leq 10^{18}$ следует, что $n \leq 60$.

Для решения задачи в варианте High потребуется несколько дополнительных утверждений.

Утверждение 1. Пусть $f(x)$ – неприводимый многочлен степени n над полем Z_p . Расширим поле Z_p новым элементом α , который введем как значение, удовлетворяющее уравнению $f(\alpha) = 0$. Новое поле $F = Z_p(\alpha)$ будет состоять из чисел вида $c_0 + c_1\alpha + c_2\alpha^2 + \dots + c_{n-1}\alpha^{n-1}$, где $c_i \in Z_p$ (поскольку α^n (а значит и более высокие степени) можно будет выразить из уравнения $f(\alpha) = 0$). Тогда справедливо:

- а) F – минимальное поле разложения многочлена $f(x)$ над Z_p , причем $f(x)$ имеет ровно n различных корней

$$\alpha, \alpha^p, \alpha^{p^2}, \dots, \alpha^{p^{n-1}}. \quad (**)$$

- б) $f(x) | x^{p^n} - x$.

Доказательство. а) Поскольку все коэффициенты a_i из поля Z_p , то по малой теореме Ферма $a_i^p = a_i$. Поэтому для любого $s \in N$ справедливы равенства

$$f(\alpha^{p^s}) = \sum_{i=0}^n a_i \cdot (\alpha^i)^{p^s} = \sum_{i=0}^n (a_i \alpha^i)^{p^s} = f(\alpha)^{p^s} = 0.$$

Предпоследнее равенство получается в силу того, что $(a+b)^p = a^p + b^p$ (все остальные биномиальные коэффициенты кратны p , когда p – простое). Отсюда следует, что, все элементы (***) являются корнями многочлена $f(x)$. Докажем, что они различны. Допустим, что $\alpha^{p^s} = \alpha^{p^t}$, где $0 \leq s < t \leq n-1$. Тогда при $r = t - s$ получаем, что $\alpha^{p^{s+r}} - \alpha^{p^s} = (\alpha^{p^r} - \alpha)^{p^s} = 0$, и значит

$$\alpha^{p^r} = \alpha, \quad 0 < r < n.$$

Отсюда будет следовать, что все p^n элементов поля F будут являться корнями многочлена $x^{p^n} - x$, что невозможно в силу условия $r < n$. Полученное противоречие доказывает, что все корни (***) различны.

б) Так как F – конечное поле и состоит из p^n элементов, то все его элементы должны удовлетворять соотношению $x^{p^n} - x = 0$. Это значит, что все корни многочлена $f(x)$ являются корнями многочлена $G(x) = x^{p^n} - x$. И значит многочлен $G(x)$ делится на $f(x)$ над полем F . Но поскольку результат такого деления является очевидно многочленом на Z_p , то $f(x)$ делит $G(x)$ и над полем Z_p .

Утверждение 2. В условиях предыдущего утверждения поле $Z_p(\alpha)$ является полем разложения любого неприводимого над Z_p многочлена $g(x)$ степени m ($m|n$), и не содержит ни одного корня неприводимого над Z_p многочлена $h(x)$, для которого $\deg h(x) \nmid n$.

Доказательство. Если $g(x)$ – неприводимый над Z_p многочлен степени m , то по утверждению 1 $g(x)|x^{p^m} - x$. Пусть $m|n$. Тогда $p^m - 1 | p^n - 1$ и значит $x^{p^m-1} - 1 | x^{p^n-1} - 1$. Следовательно, $x^{p^m} - x | x^{p^n} - x$ и $g(x)|x^{p^n} - x$. Все корни многочлена $x^{p^n} - x$ принадлежат полю $Z_p(\alpha)$, а значит $g(x)$ может иметь корни лишь из того же поля.

Если неприводимый многочлен $g(x)$ степени m имеет корень $\gamma \in Z_p(\alpha)$, то $[Z_p(\gamma) : Z_p] = m$. Тогда по теореме о башне полей, примененной к $Z_p(\alpha) \supset Z_p(\gamma) \supset Z_p$, получаем $m|n$.

Утверждение 3. При $n \in N$ справедливо равенство

$$p^n = \sum_{d|n} dI(d).$$

Доказательство. Пусть многочлен $f(x)$ неприводим над полем Z_p и $\deg f(x) = n$. По утверждению 1 поле $Z_p(\alpha)$, где α – корень многочлена

$f(x)$, является минимальным полем разложения $f(x)$ и содержит n его различных корней.

По утверждению 2 для любого неприводимого многочлена $g(x)$ над Z_p степени d , где $d|n$, поле $Z_p(\alpha)$ является полем разложения. По утверждению 1 оно содержит d различных корней многочлена $g(x)$.

Различные нормированные неприводимые многочлены не имеют общих корней в поле $Z_p(\alpha)$, так как в противном случае они бы были не взаимно просты, что для неприводимых многочленов возможно лишь при их совпадении. Значит в поле $Z_p(\alpha)$ содержится как минимум $\sum_{d|n} dI(d)$ элементов:

$$p^n \geq \sum_{d|n} dI(d).$$

С другой стороны, каждый элемент поля $Z_p(\alpha)$ является корнем многочлена $x^{p^n} - x$, и следовательно, корнем некоторого нормированного неприводимого многочлена $r(x)$. По утверждению 2 $\deg r(x)|n$ и следовательно:

$$p^n \leq \sum_{d|n} dI(d).$$

Из полученных двух неравенств следует требуемое равенство.

Утверждение 4. Если $F(n)$ и $f(n)$ – функции натурального аргумента, связанные соотношением:

$$\sum_{d|n} f(d) = F(n),$$

то при любом n имеет место формула обращения:

$$\sum_{d|n} \mu(d) F\left(\frac{n}{d}\right) = f(n).$$

Здесь $\mu(n)$ – функция Мебиуса, определяющаяся следующим образом:

$$\mu(n) = \begin{cases} 1, & \text{если } n = 1, \\ (-1)^k & \text{если } \alpha_1 = \dots = \alpha_k = 1, \\ 0 & \text{если } \exists \alpha_i > 1, \end{cases}$$

где k – количество различных простых чисел в разложении n на простые множители, а α_i – степени вхождения простых множителей в это разложение.

Доказательство этого утверждения несложно и есть практически в любом достаточно полном курсе по теории чисел (см. например Грэхем Р., Кнут Д., Паташник О. “Конкретная математика”, с.161).

Используя утверждение 3 и утверждение 4 для функций $f(n) = nI(n)$ и $F(n) = p^n$, получим формулу для количества неприводимых многочленов степени n над полем Z_p :

$$I(n) = \frac{1}{n} \sum d |n\mu(d) p^{\frac{n}{d}}.$$

При реализации вычислений по этой формуле достаточно перебирать лишь делители d числа n , свободные от квадратов. Для этого достаточно выделить все различные простые делители числа n . На это уйдет время порядка $O(\sqrt{n})$. Пусть их будет l (нетрудно проверить, что при $n \leq 10^9$ значение l не может быть больше 9). Тогда любой делитель, свободный от квадратов, будет определяться выбором одного из 2^l подмножеств этих простых делителей. Для каждого из них за $O(\log n)$ операций умножения можно вычислять значение $p^{\frac{n}{d}}$ с помощью бинарного алгоритма возведения в степень.

Следует отметить, что поскольку у нас в формуле есть деление на n , то необходимо все вычисления производить по модулю mn , а не просто m . Величина mn может достигать значения порядка 10^{18} , а это значит, что для умножения нам потребуется $O(\log mn)$ операций сложения.

Таким образом, для общего времени работы программы будет справедлива оценка $O(\sqrt{n} + 2^l \log n (\log m + \log n))$.

Задача С. До первого выпадения High

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Двое игроков играют в следующую игру. Каждый из игроков называет некоторую последовательность, состоящую из 0 и 1. После этого начинают подбрасывать симметричную монету до тех пор, пока результаты последних бросков не совпадут с последовательностью одного из игроков (значение 0 соответствует решке, 1 – гербу). Выигрывает естественно тот игрок, чья последовательность выпадет раньше. Требуется по заданным последовательностям, названным игроками, определить вероятность победы первого игрока.

Ограничения

Последовательности, названные игроками – не пусты и имеют длину не более 50. Ни одна из последовательностей не является суффиксом другой.

Формат входного файла

В первой строке содержится последовательность первого игрока, во второй строке – второго игрока (без пробелов).

Формат выходного файла

Выведите вероятность того, что последовательность первого игрока выпадет раньше, чем последовательность второго игрока с точностью не менее 10^{-8} .

Пример

stdin	stdout
001 110	0.50000000
00 10	0.25000000

Задача D. До первого выпадения Junior

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Двое игроков играют в следующую игру. Каждый из игроков называет некоторую последовательность, состоящую из 0 и 1. После этого начинают подбрасывать симметричную монету до тех пор, пока результаты последних бросков не совпадут с последовательностью одного из игроков (значение 0 соответствует решке, 1 – гербу). Выигрывает естественно тот игрок, чья последовательность выпадет раньше. Требуется по заданным последовательностям, названным игроками, определить вероятность победы первого игрока.

Ограничения

Последовательности, названные игроками – не пусты и имеют длину не более 10. Ни одна из последовательностей не является суффиксом другой.

Формат входного файла

В первой строке содержится последовательность первого игрока, во второй строке – второго игрока (без пробелов).

Формат выходного файла

Выведите вероятность того, что последовательность первого игрока выпадет раньше, чем последовательность второго игрока с точностью не менее 10^{-8} .

Пример

stdin	stdout
001 110	0.50000000
00 10	0.25000000

Разбор задачи D. До первого выпадения

Предположим сначала, что обе последовательности имеют одну и ту же длину l . Тогда мы можем быть уверены в том, что до тех пор пока не будет выполнено l подбрасываний монеты, победитель будет еще неизвестен. Кроме того, если уже сделано более, чем l бросков и никто не выиграл, то для дальнейшей игры важны лишь результаты последних l бросков. Введем обозначения: $p_{00\dots 0}$ – вероятность победы первого игрока при условии, что в последних l подбрасываниях выпадали решки, $p_{11\dots 1}$ – вероятность победы при выпадении гербов в последних l бросках, и вообще $p_{a_1 a_2 \dots a_l}$ – вероятность победы, если результат последнего броска был a_l , предпоследнего a_{l-1} и т.д.

Тогда можно записать следующие соотношения:

$$p_{a_1 a_2 \dots a_l} = \begin{cases} 1, & \text{если } a_1 a_2 \dots a_l \text{ – последовательность первого игрока,} \\ 0, & \text{если } a_1 a_2 \dots a_l \text{ – последовательность второго игрока,} \\ \frac{1}{2} (p_{a_2 \dots a_l 0} + p_{a_2 \dots a_l 1}) & \text{в остальных случаях.} \end{cases}$$

Эти соотношения образуют систему 2^l линейных уравнений с 2^l неизвестными. Их можно решить, используя например метод Гаусса. Эта система будет иметь единственное решение (исходя хотя бы из смысла задачи). Остается теперь лишь определить безусловную вероятность победы первого игрока, поскольку все возможные результаты первых l подбрасываний

несовместны и равновероятны, по формуле полной вероятности имеем

$$P = \frac{1}{2^l} \sum_{a_i \in \{0,1\}} p_{a_1 a_2 \dots a_l}.$$

Эта формула позволит кроме всего прочего сгладить возможные погрешности вычислений, которые могут возникнуть при решении системы методом Гаусса.

В случае если строка одного из игроков короче, чем у другого, можно опять же рассматривать возможные последовательности, которые могут возникнуть при первых $l = \max(l_1, l_2)$ бросках. Но тогда мы должны будем заменить условие “ $a_1 a_2 \dots a_l$ – последовательность первого (или второго) игрока” на условие “ $a_1 a_2 \dots a_l$ содержит последовательность первого (или второго) игрока в качестве подстроки” (в случае если последовательности обоих игроков содержатся в выбранной последовательности приоритет должен будет иметь игрок с более короткой последовательностью).

Вычислительную сложность алгоритма можно оценить как $O(2^{3l})$. Однако константа, которую дает метод Гаусса множителем в этой оценке, меньше единицы. И кроме того, за счет сильной разреженности матрицы системы (особенно на первых итерациях прямого хода), константа уменьшится еще как минимум в два раза.

Соответствующее замечание позволяет утверждать, что при ограничениях варианта Junior этой задачи, указанный алгоритм позволит получить результат за приемлемое время.

Перейдем к варианту High и попробуем уменьшить количество переменных. В действительности, есть смысл учитывать лишь столько последних бросков, сколько образуют максимальный по длине префикс по крайней мере одной из заданных последовательностей игроков. Различных префиксов у двух строк с длинами l_1 и l_2 может быть не более, чем $l_1 + l_2 + 1$. При ограничениях указанных в задаче это даст нам не более, чем 101 переменную. При записи уравнений соответствующие префиксы можно будет искать даже наивным алгоритмом за $O(l^2)$, что позволит построить все уравнения за $O(l^3)$. Но можно это сделать и за $O(l)$, построив бор на этих двух строках с суффиксными ссылками и переходами по каждому символу (как в алгоритме Ахо-Корасик). А именно, отождествим узел бора с префиксом, который ему соответствует. Пусть узел 0 является корнем бора (а значит соответствует пустому префиксу), -1 – фиктивный узел, не входящий в бор, $parent[x]$ – родитель узла x (соответствует префиксу на единицу меньшей длины, $next[x][c]$ – узел, в который ведет ребро бора из узла x по символу c , $last[x]$ – последний символ строки x , $suff[x]$ – суффиксная ссылка, т.е. узел (а значит соответствующий некоторому

префиксу одной из двух строк), в котором оканчивается самый длинный собственный суффикс строки x , $pr[x][c]$ – переход из узла x по символу c , т.е. узел, в котором оканчивается самый длинный (не обязательно собственный суффикс строки $x + c$). Тогда примем $pr[-1][0] = pr[-1][1] = 0$, $p[0] = -1$. Для остальных вершин (перебирая их в порядке увеличения длины строки), можно записать:

$$suff[x] = pr[suff[parent[x]]][last[x]],$$

$$pr[x][c] = \begin{cases} next[x][c], & \text{если } next[x][c] \neq -1, \\ pr[suff[x]][c] & \text{в противном случае.} \end{cases}$$

Тогда мы сможем записать уравнения в виде:

$$p_x = \frac{1}{2} (p_{pr[x][0]} + p_{pr[x][1]})$$

для всех x , не являющихся листьями в боре. Для листа вероятность определяется сразу в зависимости от того, чьей последовательности он соответствует. То, что требуется найти в задаче – это p_0 .

Теперь можем снова применить метод Гаусса для решения системы уравнений (причем за счет того, что ее матрица будет почти нижнетреугольной (т.е. в верхнем треугольнике будет ненулевым не более чем две диагонали), метод Гаусса может быть реализован с временем работы порядка $O(l^2)$. Тем не менее из-за плохой обусловленности матрицы, метод Гаусса может давать большие погрешности. В связи с этим имеет смысл представить систему в виде:

$$p = Ap,$$

для которого можно применить метод итераций в следующей форме. Поскольку $p = Ap$, можно утверждать, что при любом k выполнено $p = A^{2^k} p$. Если k будет достаточно велико, то все коэффициенты матрицы A^{2^k} будут близки к нулю, за исключением коэффициентов при переменных соответствующих листьям бора, но именно для них вероятности уже известны. Это значит, что будет легко вычислить после этого величину p_0 .

Общая сложность алгоритма составит $O(kl^3)$.

Задача Е. Раскраска забора High

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Имеется забор, состоящий из N досок, следующих одна за другой. Тре-

буется покрасить его таким образом, чтобы каждая доска была окрашена целиком в один из имеющихся C цветов. При этом некоторые пары цветов являются несовместимыми и не допускается, чтобы после покраски две соседние доски были окрашены в соответствующие цвета. Всего имеется M таких пар. Ваша задача – определить количество допустимых раскрасок, то есть таких, при которых нет двух соседних досок, покрашенных в несовместимые цвета.

Ограничения

N, C, M – целые числа.

$$1 \leq N \leq 10^{18}, 1 \leq C \leq 100, 0 \leq M \leq C(C + 1)/2.$$

Формат входного файла

В первой строке содержится три числа N, C, M . В каждой из последующих M строк содержится по два числа, определяющих пару несовместимых цветов.

Формат выходного файла

Выведите остаток от деления количества допустимых раскрасок забора на $10^9 + 7$.

Пример

stdin	stdout
2 3 2 1 2 2 3	5
3 2 1 1 2	2

Задача F. Раскраска забора Junior

Вход: stdin
Выход: stdout
Ограничение по времени: 1 с
Ограничение по памяти: 256 Мб

Имеется забор, состоящий из N досок, следующих одна за другой. Требуется покрасить его таким образом, чтобы каждая доска была окрашена целиком в один из имеющихся C цветов. При этом некоторые пары цветов являются несовместимыми и не допускается, чтобы после покраски две соседние доски были окрашены в соответствующие цвета. Всего имеется

M таких пар. Ваша задача – определить количество допустимых раскрасок, то есть таких, при которых нет двух соседних досок, покрашенных в несовместимые цвета.

Ограничения

N, C, M – целые числа.

$1 \leq N \leq 10000, 1 \leq C \leq 100, 0 \leq M \leq C(C + 1)/2$.

Формат входного файла

В первой строке содержится три числа N, C, M . В каждой из последующих M строк содержится по два числа, определяющих пару несовместимых цветов.

Формат выходного файла

Выведите остаток от деления количества допустимых раскрасок забора на $10^9 + 7$.

Пример

stdin	stdout
2 3 2 1 2 2 3	5
3 2 1 1 2	2

Разбор задачи F. Покраска забора

Построим матрицу A смежности для графа отношения совместимости, т.е. A_{ij} равно 1, если цвета i и j совместимы, и 0, если эти цвета несовместимы.

Применим метод динамического программирования. Для этого обозначим $f(n, c)$ – количество раскрасок забора длины n , для которых первая доска будет окрашена в цвет c . Тогда ответ на задачу выражается как $\sum_{c=1}^C f(N, c)$. Очевидно, что $f(1, c) = 1$. Рекуррентная формула, по которой можно определять последующие значения функции f , имеет вид:

$$f(n, c) = \sum_{d=1}^C A_{cd} f(n-1, d). \quad (*)$$

Таким образом, нам необходимо вычислить NC различных значений, для вычисления каждого из которых потребуется $O(C)$ операций, что даст общую сложность работы алгоритма порядка $O(NC^2)$.

Это вполне может укладываться в ограничения по времени варианта Junior, но явно слишком много для варианта High.

Запишем все значения $f(n, c)$ при фиксированном n в вектор-столбец f_n . Тогда формулу (*) можно будет переписать в матричной форме:

$$f_n = Af_{n-1},$$

или, используя формулу многократно, получим:

$$f_N = A^{N-1}f_1.$$

Таким образом, можно получить алгоритм со временем работы порядка $O(\log N \cdot C^3)$, если использовать бинарный алгоритм возведения в степень, и стандартный алгоритм умножение матриц за $O(C^3)$.

Задача G. Города-побратимы High

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

На координатной прямой расположены N городов. Было решено выбрать K различных пар этих городов и назвать города каждой пары городами-побратимами по отношению друг к другу. При этом, получается, что у каждого города может быть не более одного города-побратима. Требуется определить какое может быть максимальное и минимальное суммарное расстояние между городами-побратимами.

Ограничения

N, K – целые числа.
 $1 \leq N \leq 100000, 0 \leq K \leq N/2$. Координаты городов не превышают 10^9 по абсолютной величине.

Формат входного файла

В первой строке содержатся числа N и K . Во второй – N чисел, определяющих координаты городов.

Формат выходного файла

В единственной строке нужно вывести два числа – максимальное и минимальное суммарное расстояние, которое может получиться между городами-побратимами в K парах.

Пример

stdin	stdout
5 2 0 3 4 7 9	13 3
3 1 2 7 5	5 2

Задача Н. Города-побратимы Junior

Вход: stdin
Выход: stdout
Ограничение по времени: 1 с
Ограничение по памяти: 256 Мб

На координатной прямой расположены N городов. Было решено выбрать K различных пар этих городов и назвать города каждой пары городами-побратимами по отношению друг к другу. При этом, получается, что у каждого города может быть не более одного города-побратима. Требуется определить какое может быть максимальное и минимальное суммарное расстояние между городами-побратимами.

Ограничения

N, K – целые числа.

$1 \leq N \leq 10000$, $0 \leq K \leq N/2$. Координаты городов не превышают 10^9 по абсолютной величине.

Формат входного файла

В первой строке содержатся числа N и K . Во второй – N чисел, определяющих координаты городов.

Формат выходного файла

В единственной строке нужно вывести два числа – максимальное и минимальное суммарное расстояние, которое может получиться между городами-побратимами в K парах.

Пример

stdin	stdout
5 2 0 3 4 7 9	13 3
3 1 2 7 5	5 2

Разбор задачи Н. Города-побратимы

В задаче требуется найти максимальное и минимальное паросочетание, состоящее из k ребер. Конечно большое ограничение на количество вершин не позволяет нам использовать общие алгоритмы для нахождения паросочетаний. Поэтому нужно учитывать структуру графа.

Отсортируем все города по возрастанию их координат. Тогда максимальное k -паросочетание получится если взять k вершин с максимальными координатами и k вершин с минимальными координатами. Следует теперь разбить их каким-нибудь образом на пары так, чтобы в каждой паре была одна из максимальных и одна из минимальных вершин. В этом случае вес паросочетания будет равен $\sum_{N-k+1 \leq i \leq N} x_i - \sum_{1 \leq i \leq k} x_i$. Очевидно, что это наибольшее возможное значение.

Для нахождения минимального k -паросочетания заметим, что следует соединять лишь пары городов расположенных рядом друг с другом, то есть город i может быть соединен только с городом $i-1$ или $i+1$. Действительно предположим, что мы имеем оптимальное k -сочетание, в котором город i соединен с некоторым городом j ($j < i-1$). Если город $i-1$ не имеет побратима, то, поменяв пару (i, j) на пару $(i, i-1)$ мы лишь уменьшим вес паросочетания. Если же город $i-1$ занят (пусть он соединен с некоторым городом j'), то можно поменять пары (i, j) и $(i-1, j')$ на пары $(i, i-1)$ и (j, j') . Вес паросочетания при этом не увеличится. (В случае $j' > i$ имеет на самом деле смысл заменять на пары (i, j') и $(i-1, j')$, уменьшив количество пересекающихся отрезков и общий вес паросочетания. Если $j' \neq i+1$, то следующим подобным преобразованием можно будет добиться того, чтобы были связаны города i и $i+1$ без увеличения веса паросочетания).

Итак, мы выяснили, что в минимальном паросочетании могут быть лишь пары вида $(i, i+1)$. Всего таких пар будет $n-1$. И нам следует выбрать k минимальных из них, но так, чтобы ни один город не входил в несколько пар. Последнее условие не позволяет использовать жадный алгоритм, поскольку выбор ребра $(i, i+1)$ не позволяет нам выбирать на последующих шагах ребра $(i-1, i)$ и $(i+1, i+2)$.

Для решения варианта Junior, воспользуемся динамическим программированием, вычисляя значения $f(n, k)$ – минимальное k -паросочетание на городах $1, 2, \dots, n$.

Тогда $f(n, 0) = 0$, $f(n, k) = \min(f(n-1, k), f(n-2, k-1) + (x_n - x_{n-1}))$. Это позволит нам вычислить оптимальное k -паросочетание за время $O(Nk)$.

Для получения решения варианта High, усовершенствуем жадный алгоритм. Если мы выбираем ребро $(i, i+1)$, то ребра $(i-1, i)$ и $(i+1, i+2)$ не могут использоваться на последующих шагах. Единственный разумный способ использовать их, заключается в том, чтобы удалить ребро $(i, i+1)$ из паросочетания и добавить оба ребра $(i-1, i)$ и $(i+1, i+2)$, чтобы заменить старое ребро. Чтобы позволить алгоритму рассматривать такую возможность, после удаления всех указанных трех пар создадим новую виртуальную пару $(i-1, i+2)$ и присвоим ей вес $w_{i+1, i+2} + w_{i-1, i} - w_{i, i+1}$, равный величине, которая добавится к весу паросочетания, если мы воспользуемся оговоренной возможностью. Если выбранное алгоритмом ребро является первым или последним, то новую виртуальную пару создавать не понадобится.

Используя такой подход, выбирая из оставшихся пар (включая виртуальные) ту, что имеет наименьший вес – мы всегда будем получать минимальное паросочетание с количеством ребер на 1 больше. Для эффективной реализации такого алгоритма, нужно использовать структуру данных, которая позволит искать минимум, добавлять и удалять за время $O(\log N)$. Такой структурой является например куча. Единственное, о чем нужно позаботиться, – чтобы можно было для каждой пары получать следующую и предыдущую. Для этого можно хранить для каждого элемента кучи позицию в куче следующего за ним и предыдущего. Либо следует сохранять номера городов соответствующей пары, а в отдельной таблице хранить позиции (в куче) пар, в которые входит соответствующий город (их будет всегда не более двух).

Поскольку каждая операция будет выполняться за $O(\log N)$, общее время работы алгоритма будет иметь порядок $O(k \log N)$.

Задача I. Отдых у реки High

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Семейная пара решила провести отдых на берегу реки. Джордж любит

высокие места и хочет доехать туда, где берег будет как можно выше над уровнем реки. Его жена Мерри наоборот боится высоты и хочет провести отдых там, где берег будет как можно ниже. Сейчас же они едут на машине по односторонней главной дороге, вдоль которой имеется N поворотов к реке. Дорога за каждым из этих поворотов ведет к реке и каждый из супругов знает высоту, на которой расположено место, к которому ведет соответствующая дорога. За рулем разумеется Джордж, но Мерри может отвлекать Джорджа так, что тот может не заметить очередного поворота *за исключением последнего поворота* (и Джордж знает об этом). Все повороты настолько похожи, что подъезжая к очередному Джордж не может знать наверняка к какому месту он выводит. Главная дорога продолжается и за последним поворотом и тоже ведет к месту у реки с известной высотой берега. Очевидно, что Джордж может применить одну из следующих стратегий: повернуть на первом замеченном им повороте, повернуть на втором из таких поворотов, повернуть на третьем и т.д., либо вообще не сворачивать (разумеется в случае, если окажется что в действительности он заметит меньше поворотов, чем требуется в намеченной им стратегии, пара прибудет в место, расположенное в конце главной дороги).

Требуется определить оптимальную стратегию для Джорджа в предположении, что Мерри также будет действовать оптимально.

Ограничения

N, h_i – целые числа.

$1 \leq N \leq 10^5, 0 \leq h_i \leq 1000$.

Формат входного файла

В первой строке содержится число N . Во второй строке записано $N + 1$ чисел h_i , определяющих высоту места, к которому ведет дорога от i -го поворота (h_{N+1} – высота места, к которому выводит главная дорога, если никуда с нее не сворачивать).

Формат выходного файла

Выведите в первой строке максимальную среднюю высоту места, к которому может добраться Джордж, при оптимальном противодействии Мерри. Во второй строке выведите $N + 1$ число – вероятности с которыми Джордж должен применять каждую из своих чистых стратегий для достижения этой высоты. Все величины должны выводиться с точностью не менее 10^{-6} . В случае если существует несколько оптимальных стратегий, следует выбирать ту из них, у которой вероятность выбора стратегии “Нигде не сворачивать” максимальна. Если и таких несколько – ту, которая имеет наибольшую вероятность выбора N -го поворота, и т.д.

Пример

stdin	stdout
2	4.000000
0 6 3	0.333333 0.666667
	0.000000
3	2.800000
2 3 4 2	0.400000 0.400000
	0.200000 0.000000

Задача J. Отдых у реки Junior

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Семейная пара решила провести отдых на берегу реки. Джордж любит высокие места и хочет доехать туда, где берег будет как можно выше над уровнем реки. Его жена Мерри наоборот боится высоты и хочет провести отдых там, где берег будет как можно ниже. Сейчас же они едут на машине по односторонней главной дороге, вдоль которой имеется N поворотов к реке. Дорога за каждым из этих поворотов ведет к реке и каждый из супругов знает высоту, на которой расположено место, к которому ведет соответствующая дорога. За рулем разумеется Джордж, но Мерри может отвлекать Джорджа так, что тот может не заметить очередного поворота *за исключением последнего поворота* (и Джордж знает об этом). Все повороты настолько похожи, что подъезжая к очередному Джордж не может знать наверняка к какому месту он выводит. Главная дорога продолжается и за последним поворотом и тоже ведет к месту у реки с известной высотой берега. Очевидно, что Джордж может применить одну из следующих стратегий: повернуть на первом замеченном им повороте, повернуть на втором из таких поворотов, повернуть на третьем и т.д., либо вообще не сворачивать (разумеется в случае, если окажется что в действительности он заметит меньше поворотов, чем требуется в намеченной им стратегии, пара прибудет в место, расположенное в конце главной дороги).

Требуется определить оптимальную стратегию для Джорджа в предположении, что Мерри также будет действовать оптимально.

Ограничения

N, h_i – целые числа.

$$1 \leq N \leq 5000, 0 \leq h_i \leq 1000.$$

Формат входного файла

В первой строке содержится число N . Во второй строке записано $N + 1$ чисел h_i , определяющих высоту места, к которому ведет дорога от i -го поворота (h_{N+1} – высота места, к которому выводит главная дорога, если никуда с нее не сворачивать).

Формат выходного файла

Выведите в первой строке максимальную среднюю высоту места, к которому может добраться Джордж, при оптимальном противодействии Мерри. Во второй строке выведите $N + 1$ число – вероятности с которыми Джордж должен применять каждую из своих чистых стратегий для достижения этой высоты. Все величины должны выводиться с точностью не менее 10^{-6} . В случае если существует несколько оптимальных стратегий, следует выбирать ту из них, у которой вероятность выбора стратегии “Нигде не сворачивать” максимальна. Если и таких несколько – ту, которая имеет наибольшую вероятность выбора N -го поворота, и т.д.

Пример

stdin	stdout
2 0 6 3	4.000000 0.333333 0.666667 0.000000
3 2 3 4 2	2.800000 0.400000 0.400000 0.200000 0.000000

Разбор задачи J. Отдых у реки

Попытка решить сразу всю игру целиком будет не очень хорошей идеей, потому как возможных стратегий у Мерри слишком много (порядка 2^n). Попробуем разбить всю игру на последовательность более простых игр. Пусть игра G_n – это игра, заключающаяся в том, что Джордж подъезжает к перекрестку, который может иметь некоторый номер (в зависимости от выбора Мерри), но не меньший n . У Джорджа всего две стратегии – повернуть или не повернуть.

Если Мерри выбрала поворот k , а Джордж поворачивает, то результат игры сразу будет известен и равен h_k . Если же Джордж пропускает этот поворот, то супруги попадают в игру G_{k+1} .

Игра G_{N+1} — самая простая, поскольку поворотов больше нет, то сворачивать некуда, а отвлекать незачем, и потому цена V_{N+1} этой игры будет равна h_{N+1} . Решая теперь игры G_k , изменяя k от N до 1, будем получать цену игры V_k и оптимальную вероятность выбора чистой стратегии “поворачивать” p_k в такой игре.

Тогда цена всей игры будет равна V_1 , вероятность выбора стратегии “повернуть на первом замеченном повороте” — p_1 , “повернуть на втором” — $(1 - p_1)p_2$, “на третьем” — $(1 - p_1)(1 - p_2)p_3$ и т.д. Вероятность выбора стратегии “не сворачивать нигде” определится произведением $(1 - p_1)(1 - p_2) \dots (1 - p_N)$.

Для того, чтобы удовлетворить условию о максимальной вероятности выбора последних стратегий, нужно в каждой игре G_k находить минимальную оптимальную вероятность (в случае если она определяется неоднозначно).

Для решения же игры G_k , имеющей тип $2 \times (N - k + 1)$, можно использовать графический метод. Нужно построить функцию $f(p) = \min_{k \leq i \leq N} (h_i p + V_{i+1}(1 - p))$, и найти ее максимум при $p \in [0, 1]$. Эта функция является нижней огибающей для прямых $h_i p + V_{i+1}(1 - p)$. Имея эту огибающую для игры G_{k+1} ее легко будет построить для игры G_k , нужно будет пересечь очередную прямую с имеющейся огибающей и выбросить часть огибающей, расположенной над этой прямой. Используя для хранения огибающей ломаной обычный массив, это можно будет сделать за $O(l)$, где l — количество звеньев в огибающей. Тогда все решение потребует $O(N^2)$ операций, что будет достаточным для варианта задачи Junior.

В варианте High, потребуется сохранять звенья ломаной в дереве поиска (set), которое будет упорядочено по угловому коэффициенту соответствующей прямой (очевидно он будет убывать слева направо). Тогда нужно будет вставить новую прямую в это дерево, но если в нем окажется узел соответствующий прямой, которая проходит выше точки пересечения следующего и предыдущего звена, то такой узел надо будет удалить. Всего произойдет N вставок и не более чем N удалений, а так же N поисков максимума, который будет соответствовать пересечению самого дальнего звена с положительным угловым коэффициентом со следующим за ним звеном с неположительным коэффициентом. Таким образом, время работы алгоритма можно оценить величиной $O(N \log N)$.

Задача К. Количество представимых High

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Заданы два целых числа a и b . Число x назовем представимым, если его можно представить в виде суммы $x = x_1 + x_2 + \dots + x_n$ конечного (возможно нулевого) числа слагаемых x_i , каждое из которых равно a или b . Требуется определить, сколько различных чисел на отрезке $[A, B]$ являются представимыми.

Ограничения

a, b – целые числа, не превосходящие по модулю 10000.
 A, B – целые числа, не превосходящие по модулю 10^{18} .
 $A \leq B$.

Формат входного файла

В единственной строке содержатся числа a, b, A, B .

Формат выходного файла

Выведите количество представимых через a и b чисел из отрезка $[A, B]$.

Пример

<code>stdin</code>	<code>stdout</code>
4 5 7 12	4
6 10 20 30	6

Задача L. Количество представимых Junior

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Заданы два неотрицательных целых числа a и b . Число x назовем представимым, если его можно представить в виде суммы $x = x_1 + x_2 + \dots + x_n$ конечного (возможно нулевого) числа слагаемых x_i , каждое из которых равно a или b . Требуется определить, сколько различных чисел на отрезке $[A, B]$ являются представимыми.

Ограничения

a, b, A, B – целые числа.

$0 \leq a, b \leq 10000, 0 \leq A \leq B \leq 10^7$.

Формат входного файла

В единственной строке содержатся числа a, b, A, B .

Формат выходного файла

Выведите количество представимых через a и b чисел из отрезке $[A, B]$.

Пример

stdin	stdout
4 5 7 12	4
6 10 20 30	6

Разбор задачи L. Количество представимых

Требуется для заданных a и b найти числа, представимые в виде $xa + yb$, где $x, y \geq 0$. Прежде всего заметим, что нельзя получать числа, некрратные $d = \gcd(a, b)$, поскольку $xa + yb$ всегда делится на любой общий делитель a и b . Поэтому, если a и b не являются взаимно простыми, разделим их на d , а величины A и B , заменим на $\lceil A/d \rceil$ и $\lfloor B/d \rfloor$ соответственно. Теперь a и b у нас гарантированно взаимно простые. Пусть они неотрицательны (что гарантируется в варианте Junior). Тогда можно доказать, что все числа большие ab (и даже большие $ab - a - b$) представимы. Тогда можно найти за $O(ab)$ все числа, которые представимы на отрезке $[0, ab)$. И пользуясь этой информацией ответить на запрос сколько представимых чисел есть на отрезке $[A, B]$.

В варианте High, когда a и b могут быть отрицательными, нужно аккуратно разобраться со случаями. В случае, когда a и b разных знаков, все числа, кратные их наибольшему общему делителю представимы. Если же a и b имеют один знак, то представимыми могут быть лишь числа того же знака. Особого внимания требует случай $a = b = 0$, в котором единственными представимым числом является 0.

Задача М. Квадратичная перестановка High

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Рассмотрим перестановку чисел от a до b . Будем называть ее квадратичной, если для каждого элемента его сумма с элементом, который попадает на его место после перестановки, является точным квадратом. Более точно, квадратичной перестановкой называется такая биекция p множества целых чисел от a до b самого на себя, что для любого i выполняется $i + p(i) = j^2$ для некоторого целого числа j . Требуется для заданных a и b найти квадратичную перестановку.

Ограничения

a, b – целые числа.
 $0 \leq a \leq 100, 0 \leq b \leq 100000, a \leq b$.

Формат входного файла

В единственной строке содержатся числа a и b .

Формат выходного файла

Выведите $b - a + 1$ чисел, определяющих значения $p(i)$ для всех i от a до b , где p – некоторая квадратичная перестановка. Если такой перестановки при заданных a и b не существует, выведите одно число -1 .

Пример

stdin	stdout
1 9	8 2 6 5 4 3 9 1 7
3 5	-1

Задача N. Квадратичная перестановка Junior

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Рассмотрим перестановку чисел от a до b . Будем называть ее квадратичной, если для каждого элемента его сумма с элементом, который попадает

на его место после перестановки, является точным квадратом. Более точно, квадратичной перестановкой называется такая биекция p множества целых чисел от a до b самого на себя, что для любого i выполняется $i + p(i) = j^2$ для некоторого целого числа j . Требуется для заданных a и b найти квадратичную перестановку.

Ограничения

a, b – целые числа.

$0 \leq a \leq b \leq 20$.

Формат входного файла

В единственной строке содержатся числа a и b .

Формат выходного файла

Выведите $b - a + 1$ чисел, определяющих значения $p(i)$ для всех i от a до b , где p – некоторая квадратичная перестановка. Если такой перестановки при заданных a и b не существует, выведите одно число -1 .

Пример

stdin	stdout
1 9	8 2 6 5 4 3 9 1 7
3 5	-1

Разбор задачи N. Квадратичная перестановка

Задачу можно свести к задаче нахождения наибольшего паросочетания, если рассмотреть двудольный граф, в каждой из долей которого есть вершины от a до b , и соединены вершины разных долей, сумма номеров которых дает полный квадрат. Если найдется в таком графе совершенное паросочетание, то оно будет определять требуемую перестановку. Для нахождения такого паросочетания потребуется время $O(b^2\sqrt{b})$ (поскольку каждая вершина имеет не более \sqrt{b} ребер). Этого вполне достаточно, чтобы решить задачу в варианте Junior. Более того, перебор с хорошим отсечением тоже способен уложиться в ограничения по времени в этом варианте.

Для решения задачи в варианте High можно воспользоваться следующим приемом. Пусть имеется некоторое число b' такое, что $b' + 1 + b$ является точным квадратом. Тогда если найдется квадратичная перестановка для отрезка $[a, b']$, то, дополнив ее соответствиями $b' + 1 + j \rightarrow b - j$,

получим квадратичную перестановку для отрезка $[a, b]$. Эмпирическим путем можно убедиться, что при $a \leq 100$ и $b \geq 200$, такую перестановку построить всегда возможно. Таким образом, достаточно привести с помощью описанного приема (возможно применить его несколько раз) задачу к варианту, где b будет не меньше 200, но не больше скажем 500. Для такого варианта решить задачу нахождением совершенного паросочетания, после чего дополнить его до решения для исходного значения b .

Задача О. Цикл де Брёйна High

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Циклом де Брёйна порядка n для множества $D = \{0, 1, \dots, b-1\}$ называется циклическая последовательность a_0, a_1, \dots, a_{l-1} такая, что каждый вектор длины n над множеством D встречается в этой последовательности ровно один раз (т.е. для любых $b_0, b_1, \dots, b_{n-1} \in D$ существует единственное k в пределах от 0 до $l-1$ такое, что $b_j = a_{(k+j) \bmod l}$ для всех $j = \overline{0, n-1}$). Требуется построить такую последовательность.

Ограничения

n, b – целые числа.
 $1 \leq n \leq 1000, 1 \leq b \leq 10, b^n \leq 10^7$.

Формат входного файла

В единственной строке содержатся числа n и b .

Формат выходного файла

В единственной строке выведите цикл де Брёйна порядка n для множества b -ичных цифр (без пробелов).

Пример

<code>stdin</code>	<code>stdout</code>
2 3	001102122
3 2	00010111

Задача Р. Цикл де Брёйна Junior

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Циклом де Брёйна порядка n для множества $D = \{0, 1, \dots, b-1\}$ называется циклическая последовательность a_0, a_1, \dots, a_{l-1} такая, что каждый вектор длины n над множеством D встречается в этой последовательности ровно один раз (т.е. для любых $b_0, b_1, \dots, b_{n-1} \in D$ существует единственное k в пределах от 0 до $l-1$ такое, что $b_j = a_{(k+j) \bmod l}$ для всех $j = \overline{0, n-1}$). Требуется построить такую последовательность.

Ограничения

n, b – целые числа.
 $1 \leq n \leq 1000, 1 \leq b \leq 10, b^n \leq 10^4$.

Формат входного файла

В единственной строке содержатся числа n и b .

Формат выходного файла

В единственной строке выведите цикл де Брёйна порядка n для множества b -ичных цифр (без пробелов).

Пример

stdin	stdout
2 3	001102122
3 2	00010111

Разбор задачи Р. Цикл де Брёйна

Построим граф де Брёйна – ориентированный граф с вершинами, соответствующими различным векторам длины n с элементами из D , в котором из вершины (x_1, \dots, x_n) в вершину (y_1, \dots, y_n) ведет ребро в том и только в том случае, когда $y_{i+1} = x_i$ для $i = \overline{1, n-1}$. При этом самому ребру можно сопоставить набор длины $n+1$: $(x_1, \dots, x_n, y_n) = (x_1, y_1, \dots, y_n)$. Тогда задача сводится к тому, чтобы построить в этом графе цикл, который проходит через каждую вершину ровно один раз. Как известно, такие циклы называются гамильтоновыми и для их поиска в общем случае неизвестны полиномиальные алгоритмы. Но обратим внимание на ребра. Если

мы пройдем по каждому ребру по одному разу, то мы получим цикл де Брёйна порядка $n + 1$. А такие циклы, которые называются эйлеровыми, можно строить за время сравнимое с количеством ребер в графе. В силу специфики графа, он будет связным и входящая степень каждой вершины равна исходящей, поэтому такой цикл всегда будет существовать. Таким образом, для построения цикла де Брёйна нужно будет построить граф де Брёйна для векторов длины $n - 1$ и найти в нем эйлеров цикл. Выбирая первый элемент вектора соответствующего каждому ребру этого цикла мы получим цикл де Брёйна. В варианте Junior такой граф может быть явно построен и представлен в виде матрицы смежности, а варианте High соответствующий граф нужно будет представлять по крайней мере списками смежности, а лучше и вовсе не строить в явном виде, а получать ребра по мере достижения вершин. Сложность алгоритма – $O(b^n)$.

Задача Q. Карточный поединок High

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Двое игроков играют в игру. Каждый из игроков изначально получает некоторое количество карточек (пусть будет n_1 и n_2 соответственно). Каждый ход игроки выбирают по одной из имеющихся у них на руках карточек, и затем открывают их. Более слабая карта сбрасывается в отбой, а более сильную игрок, показавший ее, забирает обратно. В случае, если игроки показали одинаковые карты, они обе сбрасываются в отбой. Игра продолжается до тех пор, пока хотя бы у одного из игроков не закончатся карты. Если при этом у одного из игроков осталась еще хотя бы одна карта, он получает 1 очко, а соперник – 0. Если же у обоих игроков закончились карты, каждый получает по 0.5 очков. Всего есть N типов карт. Отношение силы карт может быть нетранзитивным и задается матрицей A . A_{ij} равно 1, если карта i бьет карту j , и 0 в противном случае. Требуется определить цену этой игры для первого игрока в предположении, что второй игрок играет оптимально.

Ограничения

$1 \leq n_1, n_2, N \leq 8$.
 $A_{ij} + A_{ji} = 1$ при $i \neq j$, $A_{ii} = 0$.

Формат входного файла

В первой строке задается число N . Последующие N строк содержат по N чисел, определяющих матрицу A . Следующая строка содержит число n_1 и еще n_1 чисел, каждое из которых определяет тип соответствующей карточки первого игрока. В последней строке в аналогичном формате задаются карточки второго игрока.

Формат выходного файла

Выведите цену игры для первого игрока с точностью не менее 10^{-8} .

Пример

stdin	stdout
3 0 1 1 0 0 1 0 0 0 2 3 2 1 1	0.00000000
3 0 1 0 0 0 1 1 0 0 3 1 2 3 3 1 2 3	0.50000000
3 0 1 0 0 0 1 1 0 0 3 1 2 3 3 2 2 3	0.66666667

Задача R. Карточный поединок Junior

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Двое игроков играют в игру. Каждый из игроков изначально получает некоторое количество карточек (пусть будет n_1 и n_2 соответственно).

Каждый ход игроки выбирают по одной из имеющихся у них на руках карточек, и затем открывают их. Более слабая карта сбрасывается в отбой, а более сильную игрок, показавший ее, забирает обратно. В случае, если игроки показали одинаковые карты, они обе сбрасываются в отбой. Игра продолжается до тех пор, пока хотя бы у одного из игроков не закончатся карты. Если при этом у одного из игроков осталась еще хотя бы одна карта, он получает 1 очко, а соперник – 0. Если же у обоих игроков закончились карты, каждый получает по 0.5 очков. Всего есть N типов карт. Отношение силы карт может быть нетранзитивным и задается матрицей A . A_{ij} равно 1, если карта i бьет карту j , и 0 в противном случае. Требуется определить максимальный средний выигрыш игрока в предположении, что второй игрок использует равномерную стратегию (то есть каждый ход выбирает одну из своих карт случайно с равной вероятностью).

Ограничения

$$1 \leq n_1, n_2, N \leq 8.$$

$$A_{ij} + A_{ji} = 1 \text{ при } i \neq j, A_{ii} = 0.$$

Формат входного файла

В первой строке задается число N . Последующие N строк содержат по N чисел, определяющих матрицу A . Следующая строка содержит число n_1 и еще n_1 чисел, каждое из которых определяет тип соответствующей карточки первого игрока. В последней строке в аналогичном формате задаются карточки второго игрока.

Формат выходного файла

Выведите цену игры для первого игрока с точностью не менее 10^{-8} .

Пример

stdin	stdout
3 0 1 1 0 0 1 0 0 0 2 3 2 1 1	0.00000000
3 0 1 0 0 0 1 1 0 0 3 1 2 3 3 1 2 3	0.66666667
3 0 1 0 0 0 1 1 0 0 3 1 2 3 3 2 2 3	0.66666667

Разбор задачи R. Карточный поединок

Применим динамическое программирование для решения этой задачи. Пусть S_1 – множество (точнее говоря мультимножество) карт, которые остались у первого игрока, S_2 – множество карт, которые остались у второго игрока. Разумеется следует рассматривать только такие множества, которые являются подмножествами исходных множеств карт. Пусть $V(S_1, S_2)$ – цена игры с соответствующими множествами карт у игроков. У каждого игрока есть возможные стратегии выбрать одну из своих карт и сделать ход. В результате мы попадаем в состояние (S'_1, S'_2) , где $S'_i = S_i$, если карта i -го игрока бьет карту соперника, и $S'_i = S_i \setminus j$, если выбранная i -ым игроком карта j слабее карты соперника или совпадает с ней. Предположим, что игры для всех подмножеств S_1 и S_2 уже решены, тогда построим матрицу игры, в которой значениями будут цены игр на соответствующих подмножествах, и решим ее. В варианте Junior мы знаем стратегию второго игрока, поэтому нам достаточно будет выбрать чистую стратегию, приносящую максимальный выигрыш. В варианте High нужно искать смешанную стратегию, которая принесет максимальный выигрыш, для чего можно будет записать соответствующую задачу линейного про-

граммирования и решить ее симплекс-методом. Начальные значения здесь $V(\emptyset, \emptyset) = 0.5$, $V(\emptyset, S) = 0$, $V(S, \emptyset) = 1$ для любого непустого множества S . И потребуется найти значение $V(S_1^0, S_2^0)$, где S_i^0 – множество карт i -го игрока в начале игры. Среднее время работы такого решения можно будет оценить величиной порядка $O(2^{n_1+n_2}N^3)$.

Задача S. Посадка в самолет High

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

N человек садятся в самолет, в котором есть в точности N мест. У каждого из пассажиров есть билет на некоторое место в этом самолете и нет двух одинаковых билетов. Однако некоторые из пассажиров являются сумасшедшими. Люди заходят по одному в салон самолета. Сумасшедшие люди при входе в самолет не смотрят на билет, а садятся на некоторое место, выбирая его равновероятно из свободных, нормальные же люди занимают место, указанное в билете. Но если место нормального пассажира занято, то чтобы не начинать скандал, он садится на любое свободное место с равной вероятностью. Требуется определить для каждого пассажира вероятность того, что он займет при входе в самолет место, указанное в его билете.

Ограничения

N – целое число, $1 \leq N \leq 100000$.

Формат входного файла

В первой строке задается число N . Во второй строке записаны N чисел, каждое из которых определяет соответствующего пассажира в порядке входа в салон самолета (0 обозначает нормального человека, 1 – сумасшедшего).

Формат выходного файла

Выведите N чисел, каждое из которых определяет вероятность того, что соответствующий пассажир займет свое место. Все значения должны быть выведены с точностью не меньше 10^{-8} .

Пример

stdin	stdout
3 0 1 0	1.00000000 0.50000000 0.50000000
4 1 0 1 0	0.25000000 0.75000000 0.33333333 0.33333333

Задача Т. Посадка в самолет Junior

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

N человек садятся в самолет, в котором есть в точности N мест. У каждого из пассажиров есть билет на некоторое место в этом самолете и нет двух одинаковых билетов. Однако некоторые из пассажиров являются сумасшедшими. Люди заходят по одному в салон самолета. Сумасшедшие люди при входе в самолет не смотрят на билет, а садятся на некоторое место, выбирая его равновероятно из свободных, нормальные же люди занимают место, указанное в билете. Но если место нормального пассажира занято, то чтобы не начинать скандал, он садится на любое свободное место с равной вероятностью. Требуется определить для каждого пассажира вероятность того, что он займет при входе в самолет место, указанное в его билете.

Ограничения

N – целое число, $1 \leq N \leq 15$.

Формат входного файла

В первой строке задается число N . Во второй строке записаны N чисел, каждое из которых определяет соответствующего пассажира в порядке входа в салон самолета (0 обозначает нормального человека, 1 – сумасшедшего).

Формат выходного файла

Выведите N чисел, каждое из которых определяет вероятность того, что соответствующий пассажир займет свое место. Все значения должны быть выведены с точностью не меньше 10^{-8} .

Пример

stdin	stdout
3 0 1 0	1.00000000 0.50000000 0.50000000
4 1 0 1 0	0.25000000 0.75000000 0.33333333 0.33333333

Разбор задачи Т. Посадка в самолет

Можно попытаться решить эту задачу с помощью динамического программирования, по крайней мере для варианта Junior. Пусть $p(i, a_1 a_2 \dots a_n)$ – вероятность того, что i -ый пассажир займет свое место при условии, что у нас есть n пассажиров, и первый пассажир – обычный и его место не занято, если $a_1 = 0$; сумасшедший и его место не занято, если $a_1 = 1$; не имеет значения какой, но его место занято, если $a_1 = 2$ (аналогично описывается состояние и остальных пассажиров). Тогда можно записать начальные значения:

$$p(1, a_1 a_2 \dots a_n) = \begin{cases} 1, & \text{если } a_1 = 0, \\ 1/n, & \text{если } a_1 = 1, \\ 0, & \text{если } a_1 = 2. \end{cases}$$

Для остальных пассажиров верна рекуррентная формула:

$$p(i, a_1 a_2 \dots a_n) = \begin{cases} p(i-1, a_2 \dots a_n), & \text{если } a_1 = 0, \\ \frac{1}{n-k} (p(i-1, a_2 \dots a_n) + \sum p(i-1, a'_2 \dots a'_n)), & \text{если } a_1 = 1, \\ \frac{1}{n-k} (\sum p(i-1, a'_2 \dots a'_n)), & \text{если } a_1 = 2. \end{cases}$$

Здесь k – количество элементов a_j равных 2, $a'_2 \dots a'_n$ – наборы, получающиеся при замене одного из элементов не равных 2 этим самым значением. Общая сложность работы такого алгоритма можно оценить величиной порядка $O(n^2 3^n)$, но разумеется это очень пессимистичная оценка, поскольку далеко не все 3^n последовательностей $a_1 a_2 \dots a_n$ возможны.

Для решения варианта High, на нескольких примерах можно заметить, что $p(i, a_1 a_2 \dots a_n) = \frac{a_1 ? 1 : (n - i + 1)}{n - i + 1 - \sum_{1 \leq k < i} a_k}$ при условии что все a_k отличны

от 2. Эту формулу можно доказать индукцией по количеству человек. Решение, использующее эту формулу, будет иметь асимптотику $O(n)$.

День пятый (20.02.2013 г.)

Контест Геральда Агапова и Фефера Ивана

Об авторах...

Геральд Агапов, учится в Саратовском Государственном Университете, 5 курс, факультет Компьютерных Наук и Информационных Технологий.

Рейтинг TopCoder: 2309

Рейтинг Codeforces: 2275

С февраля 2012 года является координатором задач Codeforces.

Участвовал в подготовке VK Cup 2012.



Фефер Иван, учится в Саратовском Государственном Университете, 5 курс, факультет Компьютерных Наук и Информационных Технологий.

Рейтинг TopCoder: 1851

Рейтинг Codeforces: 2145

Участвовал в онсайт-раундах VK Cup 2012 и Russian Code Cup 2012.



Основные достижения авторов :

- Участвуют в Петрозаводских Сборах с 2010 года. На Зимних Сборах 2013 заняли 6 место.
- Участвовали в турнире ICL (г. Казань) с 2010 года. 2010 год - 5 место, 2011 год - 3 место, 2012 год - 2 место.
- Участвуют в NEERC с 2010 года. В 2011 году заняли 7 место.

- По результатам голосования Best Of 2011 на сайте snarknews.info команда Saratov SU #1 победила в номинации "Прогресс года 2011".

Теоретический материал. Топологическая сортировка онлайн

Рассказанная лекция целиком взята из статьи:

<http://www.cs.princeton.edu/%7Esssix/papers/dto-journal.pdf>.

Задачи и разборы

Задача А. Обнаружение циклов

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вам дан ориентированный граф, состоящий из n вершин. Изначально этот граф не содержит ребер. Также вам даны m запросов на добавление ребра в граф.

Пронумеруем вершины графа целыми числами от 1 до n . Ребро, ведущее из вершины x в вершину y , будем обозначать (x, y) . Рассмотрим запрос на добавление ребра (v_i, u_i) . В ответ на запрос Вам нужно сделать следующее:

- Добавить ребро (v_i, u_i) в граф.
- Если после добавления этого ребра в графе появляется цикл, нужно вывести «-», после чего удалить из графа только что добавленное ребро (v_i, u_i) .
- Если после добавления этого ребра граф остается ациклическим, нужно вывести «+».

Выполните все заданные запросы.

Формат входного файла

В первой строке заданы два целых числа n и m ($1 \leq n \leq 10^4$; $1 \leq m \leq 3 \cdot 10^5$). В следующих m строках заданы запросы. В i -той строке записаны два целых числа v_i, u_i ($1 \leq v_i, u_i \leq n$) — запрос на добавление ребра (v_i, u_i) .

Обратите внимание, в граф могут добавляться кратные ребра и петли. Гарантируется, что все **запросы генерируются случайным образом**. Добавляемое ребро выбирается равновероятно среди всех возможных ребер.

Формат выходного файла

Выведите m строк — ответы на запросы в порядке следования запросов во входных данных.

Пример

stdin	stdout
3 7	+
1 2	+
1 2	−
1 1	+
2 3	−
3 1	−
3 1	+
1 3	
1 3	−
1 1	−
1 1	−
1 1	
4 6	+
1 2	+
1 3	+
1 4	+
2 3	−
4 1	+
3 4	

Разбор задачи А. Обнаружение циклов

В этой задаче нужно было реализовать алгоритм поддержания топологической сортировки в online за $O(|V| \cdot |E|)$. Тесты были сгенерированы случайным образом, поэтому такое решение работало достаточно быстро, чтобы пройти ограничения по времени. Единственное существенное, что следует сказать, что при реализации этого алгоритма надо было аккуратно находить цикл в графе, если он вдруг появлялся. А именно, в первоначальном варианте алгоритма граф ациклический — все инварианты алгоритма выполняются только для ациклического графа. Инварианты будут сохраняться, если в алгоритме поиска цикла прекращать поиск цикла в момент

нахождения самого первого цикла. Например, пусть мы выполняем запрос на добавление ребра (v, u) , если писать реализацию алгоритма, используя поиск в глубину, то нужно выходить из всех уровней рекурсии сразу, если обход в глубину посетил вершину v .

Задача В. Топологическая сортировка

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вам дан ориентированный граф, состоящий из n вершин. Каждая вершина графа имеет свой цвет. Изначально этот граф не содержит ребер. Также вам даны m запросов на добавление ребра в граф.

Пронумеруем вершины графа целыми числами от 1 до n . Цвет вершины с номером v обозначим c_v . Изначально цвет каждой вершины совпадает с ее номером, то есть $c_v = v$ для всех вершин. Ребро, ведущее из вершины x в вершину y , будем обозначать (x, y) . Рассмотрим запрос на добавление ребра (v_i, u_i) . В ответ на запрос Вам нужно сделать следующее:

- Добавить ребро (v_i, u_i) в граф.
- Найти **лексикографически наименьшую по последовательности цветов** вершин графа топологическую сортировку графа. То есть найти такую лексикографически наименьшую по последовательности цветов перестановку p_1, p_2, \dots, p_n , что для любого ребра графа (v, u) выполняется неравенство $x_v < x_u$, где $p_{x_v} = v$, а $p_{x_u} = u$.
- Вывести полиномиальный хеш от найденной топологической сортировки. То есть вывести остаток от деления величины $\sum_{i=1}^n p_i \cdot (10^6 + 3)^{(i-1)}$ на $1000000009 (10^9 + 9)$.
- Перекрасить вершины графа в соответствии с топологической сортировкой графа. То есть для всех вершин v выполнить присвоение $c_v = j$, где $p_j = v$.

Выполните все заданные запросы.

Формат входного файла

В первой строке заданы два целых числа n и m ($1 \leq n \leq 10^3$; $1 \leq m \leq 50000$). В следующих m строках заданы запросы. В i -той строке записаны

два целых числа v_i, u_i ($1 \leq v_i, u_i \leq n$; $v_i \neq u_i$) — запрос на добавление ребра (v_i, u_i) .

Обратите внимание, в граф могут добавляться кратные ребра. Гарантируется, что граф никогда не становится циклическим.

Формат выходного файла

Для каждого запроса выведите в отдельной строке полиномиальный хеш от найденной топологической сортировки по модулю 1000000009 ($10^9 + 9$).

Пример

stdin	stdout
3 3	8991020
3 1	8991020
2 1	7991018
3 2	
5 4	350074640
5 1	276308329
5 2	265326315
5 3	264326313
5 4	

Примечание

Перестановкой, состоящей из n элементов, называется последовательность n различных целых чисел, каждое из которых не меньше 1 и не больше n .

Последовательность x_1, x_2, \dots, x_p *лексикографически меньше* последовательности y_1, y_2, \dots, y_q , если либо $p < q$ и $x_1 = y_1, x_2 = y_2, \dots, x_p = y_p$, либо существует такое число r ($r < p, r < q$), что $x_1 = y_1, x_2 = y_2, \dots, x_r = y_r$ и $x_{r+1} < y_{r+1}$.

Топологические сортировки после каждого шага для первого тестового примера:

- 2, 3, 1;
- 2, 3, 1;
- 3, 2, 1.

Разбор задачи В. Топологическая сортировка

В этой задаче нужно было реализовать алгоритм поддержания топологической сортировки в online за $O(|V| \cdot |E|)$. Тесты здесь не были сгенерированы случайным образом, поэтому ограничения были значительно меньше, чем в задаче “Обнаружение циклов”. Дополнительно, в задаче требовалось реализовать полиномиальное хеширование так, чтобы можно было быстро (за $O(1)$) пересчитывать значение хеш-функции для меняющейся последовательности. Последнее можно сделать следующим образом:

1. Завести массивы $x[i] = 1000003^i$, $p[i]$ — массив, который хранит топологическую сортировку.
2. Посчитать один раз за линейное время значение $res = \sum_{i=1}^{i=n} x[i]^{i-1} \cdot p[i]$ по заданному модулю.
3. Пусть меняется значение в ячейке j массива p , нужно пересчитать res . Например, пусть нужно выполнить $p[j] = val$. Тогда, нужно выполнить последовательность операций пересчета res : $res = res - (x[j-1] \cdot p[j]); p[j] = val; res = res + (x[j-1] \cdot p[j])$.

Задача С. Расписание на дереве

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дан один станок и множество, состоящее из n работ, причем для каждой работы задано ее время выполнения p_j и вес w_j . Также между работами заданы зависимости такие, что каждая из некоторых $n - 1$ работ зависит ровно от одной другой работы (для разных работ работы, от которых они зависят, могут различаться), а одна работа — не зависит ни от одной. Таким образом зависимости образуют ориентированное корневое дерево.

Выполните заданные работы на одном станке. Необходимо минимизировать взвешенную сумму времен завершения работ $\sum w_j C_j$.

В одно и тоже время на станке не может выполняться более одной работы. Также нельзя прерываться во время выполнения какой-то работы, нужно выполнять ее до конца.

Формат входного файла

В первой строке дано целое число n ($1 \leq n \leq 50000$) — количество работ. Во второй строке даны n целых чисел p_j ($1 \leq p_j \leq 1000$) — времена выполнения работ. В третьей строке даны n целых чисел w_j ($1 \leq w_j \leq 1000$) — веса работ.

В следующих $n - 1$ строках даны пары целых чисел v_i и u_i ($1 \leq v_i, u_i \leq n$) — пара означает, что перед выполнением работы u_i необходимо выполнить работу v_i . Гарантируется, что зависимости образуют ориентированное корневое дерево.

Формат выходного файла

В первой строке выведите целое число — оптимальное значение целевой функции. В следующей строке выведите произвольное оптимальное расписание — n целых чисел t_1, t_2, \dots, t_n . Где t_i — момент времени, в который нужно начать выполнять i -ю работу.

Пример

stdin	stdout
3 1 3 2 1 6 4 1 2 1 3	49 0 3 1
4 3 4 2 1 2 3 3 2 2 1 3 4 2 3	64 7 0 4 6
7 1 2 3 4 5 6 7 7 6 5 4 3 2 1 1 2 1 3 2 4 2 5 3 6 3 7	210 0 1 3 6 10 15 21

Разбор задачи С. Расписание на дереве

Условие. У каждой задачи i есть время выполнения t_i и некоторый вес w_i . Есть корневое дерево зависимостей. Чтобы выполнить задачу, нужно сперва выполнить ее отца. Корень дерева можно выполнить сразу. Нужно выполнить все задачи, минимизируя величину $\sum w_i T_i$, где T_i — время начала выполнения i -й задачи.

Решение. Представим, что никакого дерева нет. Задачи можно выполнять в любом порядке. Пусть $n = 2$, тогда, если мы выполним задачи в порядке $(1, 2)$, получится штраф $t_1 w_2$, иначе $t_2 w_1$. Получаем, что, если $\frac{t_1}{w_1} < \frac{t_2}{w_2}$, то задачи выгодно выполнять в порядке $(1, 2)$, иначе в порядке $(2, 1)$. Получили, что для любого n любые две подряд выполняемые задачи обладают свойством: $\frac{t_1}{w_1} < \frac{t_2}{w_2}$, иначе их можно поменять местами, и ответ улучшится. Если дерева нет, оптимально выполнять задачи в порядке возрастания величины $\frac{t_i}{w_i}$.

Теперь вернемся к исходной задаче. Дерево есть. Выберем задачу с минимальным $\frac{t_i}{w_i}$, как только мы выполнили ее отца в дереве, выгодно сразу же выполнить данную задачу. Доказательство: пусть у нас есть возможность ее выполнить сразу, но мы выполним ее не сразу, тогда, можно, сдвинув ее в порядке выполнения ближе к началу, улучшить ответ.

Получили, что мы выполним сперва отца вершины p_v , а затем сразу v . Значит, их можно сжать в одну задачу. Время выполнения новой задачи равно $t_{p_v} + t_v$, вес соответствующий штрафу равен $w_{p_v} + w_v$. Сжимать вершины будем с помощью системы непересекающихся множеств. Параметры t и w будем хранить в «корне множества». Также в корне множества будем хранить список задач, соответствующих множеству, в том порядке, в котором их нужно выполнить. Процесс сжатия вершины и ее отца нужно повторять до тех пор, пока не останется только корень. После этого в корне будет храниться список с порядком выполнения всех задач. В каждый момент времени нужно быстро выбирать вершину дерева с минимальным $\frac{t_i}{w_i}$, для этого все вершины кроме корня храним в **set (TreeSet)**. Достав вершину, и объединив ее с отцом, удаляем и вершину, и отца из **set** и, если получившееся объединение не является корнем, добавляем его в **set**.

Задача D. Братья по крови наносят ответный удар

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Поликарп раздобыл дерево родственных связей. Найденное дерево описывает родственные связи n человек, пронумерованных от 1 до n . Каждый человек в этом дереве имеет не более одного непосредственного предка. Также каждый человек в дереве имеет коня некоторого типа, человек номер x имеет коня типа t_x . Типы коней не обязательно уникальные.

Назовем человека с номером a 1-предком человека с номером b , если человек с номером a является непосредственным предком человека с номером b .

Назовем человека с номером a k -предком ($k > 1$) человека с номером b , если у человека с номером b есть 1-предок, и человек с номером a является $(k - 1)$ -предком 1-предка человека с номером b .

В найденном дереве родственные связи не образуют циклов. Другими словами не существует человека, который непосредственно или косвенно является собственным предком (то есть является x -предком самого себя, для некоторого x , $x > 0$).

Назовем человека с номером a k -сыном человека с номером b , если человек с номером b является k -предком человека с номером a .

Поликарп очень сильно интересуется сколько у кого и каких коней. Он записал на листочке m пар чисел v_i, k_i . Помогите ему для каждой пары v_i, k_i узнать величину *конности* множества k_i -сыновей человека с номером v_i .

Конностью множества людей p_1, p_2, \dots, p_r ($p_1 < p_2 < \dots < p_r$) называется число $(t_{p_1} \rightarrow (t_{p_2} \rightarrow (\dots \rightarrow t_{p_r})))$. *Конность* пустого множества равно нулю. Здесь операция $x \rightarrow y$ обозначает операцию побитового **СЛЕДОВАТЕЛЬНО** двух 30-битных чисел. Подробное описание операции смотрите в примечании.

Формат входного файла

В первой строке входных данных записано единственное целое число n ($1 \leq n \leq 10^5$) — количество людей в дереве. В следующих n строках записано описание людей в дереве. В i -той из этих строк записаны через пробел целое число t_i и целое число r_i ($0 \leq t_i \leq 10^9, 0 \leq r_i \leq n$), где t_i — тип коня человека с номером i , а r_i — номер непосредственного предка человека с номером i или 0 если у человека с номером i нет непосредственного предка.

В следующей строке записано единственное целое число m ($1 \leq m \leq 10^5$) — количество записей Поликарпа. В m следующих строках записаны пары целых чисел через пробел. В i -ой строке записаны целые числа v_i, k_i ($1 \leq v_i, k_i \leq n$).

Гарантируется, что родственные связи не образуют циклов.

Формат выходного файла

Выведите m целых чисел, разделенных пробельными символами, — ответы на записи Поликарпа. Ответы для записей выводите в том порядке, в котором записи встречаются во входных данных.

Пример

stdin	stdout
6	1073741822
1 0	1073741823
1 1	1073741823
0 1	0
3 2	
4 2	
5 2	
4	
1 1	
1 2	
2 1	
3 1	

Примечание

Результат операции побитового СЛЕДОВАТЕЛЬНО двух 30-битных чисел — это 30-битное число, каждый из 30 бит которого получается операцией битового следовательно двух битов операндов. В частности $10 \rightarrow 132 = 1073741813$.

Таблица истинности для операции следовательно для битов:

- $0 \rightarrow 0 = 1$;
- $0 \rightarrow 1 = 1$;
- $1 \rightarrow 0 = 0$;
- $1 \rightarrow 1 = 1$.

Разбор задачи D. Братья по крови наносят ответный удар

Решение задачи работает за $O(n \cdot \sqrt{n} \cdot \log n)$. Будем считать, что решаем задачу для одного дерева. Для нескольких компонент связности деревьев решение обобщается очевидным образом.

1. Преподсчитаем для каждой вершины ее глубину $dep(v)$ — расстояние в ребрах до корня дерева.
2. Разобьем вершины на группы по величине $dep(v)$, в каждой группе отсортировав вершины в порядке посещения их обходом в глубину.
3. Пусть приходит запрос v, k . Тогда очевидно, что этот запрос является некоторым запросом на отрезке в группе, соответствующей глубине $dep(v) + k$.
4. Выполним запрос на отрезке в этой группе за длину отрезка умноженную на логарифм. После чего закешируем ответ на запрос в хеш-таблице.
5. Если в следующий раз нам придется выполнять точно такой же запрос на точно таком же отрезке группы $dep(v) + k$, возьмем уже подсчитанное значение ответа на запрос из хеш-таблицы.
6. Суммарная длина всех отрезков, на которых мы будем считать ответы на запросы (не за $O(1)$, доставая значение из хеш-таблицы) будет не более чем $O(n \cdot \sqrt{n})$.

Задача E. Гиперраздление

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вам даны n точек. Каждая точка из пространства R^d .

Нужно разделить их $d - 1$ мерной плоскостью, проходящей через точку $(0, 0, \dots, 0)$. Гарантируется, что решение всегда существует.

Формат входного файла

В первой строке заданы два целых числа n ($1 \leq n \leq 5000$), d ($1 \leq d \leq 10$).

Далее n строк содержат по $d+1$ чисел. Первые d чисел — вещественные числа, координаты очередной точки $x_{j,1}, x_{j,2}, \dots, x_{j,d}$. Последнее число — целое число y_j ($|y_j| = 1$). Число y_j определяет положение точки относительно плоскости, читайте дальше для более точного понимания.

Координаты точек по модулю не превосходят 100. Все вещественные числа содержат не более 6 знаков после десятичной точки.

Формат выходного файла

Выведите d вещественных чисел n_1, n_2, \dots, n_d — нормаль плоскости. Нормаль — вектор произвольной **положительной** длины. Для каждой точки $\text{sign}(\sum_{i=1}^d x_{j,i} \cdot n_i)$ должен быть равен ее y_j . Если решений несколько, можно вывести любое.

Ответ будет считаться корректным, если для каждой точки $|\sum_{i=1}^d x_{j,i} \cdot n_i| \geq 10^{-4}|n|$, где $|n|$ — длина вектора нормали.

Пример

stdin	stdout
2 1 1 -1 -1 1	-1.0000000000
4 3 -99.749 12.71 -61.33 1 61.7 17.00 -4.0 -1 -29.94 79.192 64.56 1 49.320 -65.178 71.788 -1	-0.891651465 0.415648308 -0.179427280

Разбор задачи Е. Гиперразделение

Условие. В d -мерном пространстве даны n точек. Для каждой точки указано, по какую сторону от гиперплоскости она должна лежать. Нужно провести через начало координат гиперплоскость так, чтобы каждая из точек лежала там, где нужно. Гарантируется, что ответ существует.

Решение. Плоскость проходящая через начало координат задается только вектором нормали r . То, с какой стороны от плоскости лежит точка, задается знаком скалярного произведения с нормалью. Те точки, у которых знак должен быть отрицательным, домножим на -1 . Теперь задача формулируется так: выбрать вектор r так, чтобы все скалярные произведения $\langle r, x_i \rangle$ были положительны. Это можно сделать так:
 $r_0 = 0$, пока существует $x_j : \langle r, x_j \rangle \leq 0$ делаем $r_{i+1} = r_i + x_j$


```
1 read(n)
2 for (i = 1; i <= n; i++)
3   read(x[i], sign[i])
4   x[i] *= sign[i]
5 r = 0
6 run = 1
7 while (run)
8   run = 0
9   for (i = 1; i <= n; i++)
10     if (r * x[i] <= 0)
11       r += x[i]
12 output(r)
```

Доказательство сходимости.

Докажем, что процесс конечен, и оценим число шагов. Ответ существует, обозначим его за u . Нам важно только направление нормали, поэтому предположим, что $|u| = 1$. Длина проекции на единичный вектор равна величине скалярного произведения с ним. Оценим длину проекции $\langle r_k, u \rangle$. $\forall i : \langle x_i, u \rangle > 0$, обозначим $\min_i \langle x_i, u \rangle$ за α , тогда $\langle r_k, u \rangle = \langle r_{k-1}, u \rangle + \langle x_j, u \rangle \geq \langle r_{k-1}, u \rangle + \alpha$, значит $\langle r_k, u \rangle \geq k \cdot \alpha$. Возведем обе части в квадрат и воспользуемся неравенством Коши-Буняковского. Получим $|r_k|^2 = |r_k|^2 |u|^2 \geq (\langle r_k, u \rangle)^2 = (k \cdot \alpha)^2$. Теперь получим оценку сверху $|r_k|^2 = |r_{k-1} + x_j|^2 = |r_{k-1}|^2 + 2\langle r_{k-1}, x_j \rangle + |x_j|^2$. По построению $\langle r_{k-1}, x_j \rangle \leq 0$, поэтому $|r_k|^2 \leq |r_{k-1}|^2 + |x_j|^2$. Обозначим $\max_i |x_i|^2$ за β , тогда $|r_k|^2 \leq k \cdot \beta$. Соберем накопленные знания: $(k \cdot \alpha)^2 \leq |r_k|^2 \leq k \cdot \beta$. Упростим формулу, отнормировав все x_i , тогда $|x_i| = 1$, значит $\beta = 1$, а $\alpha = \cos t$, где $t \in [0.. \pi)$ — половина угла раствора минимального описывающего конуса.

$$(k \cdot \alpha)^2 \leq |r_k|^2 \leq k$$

Процесс точно сойдется, когда $(k \cdot \alpha)^2 = k$, поэтому $k \leq \alpha^{-2}$.

День шестой (21.02.2013 г.)

Контеcт Фефера Ивана и Геральда Агапова

Теоретический материал. Суффиксное дерево

Основные понятия

Бор (или *префиксное дерево*) — это такое дерево, построенное для набора строк, которое содержит все префиксы этого набора строк. Данное определение не очень формальное, вы можете прочитать более подробное описание бора на сайте http://ru.wikipedia.org/wiki/Префиксное_дерево. Сжатый бор — это бор, который занимает линейное количество памяти от количества строк в множестве, если считать размер алфавита константой.

Суффиксное дерево — это сжатый бор, построенный на множестве суффиксов заданной строки. Суффиксное дерево называется *неявным*, если оно содержит наименьшее количество вершин, среди всех суффиксных деревьев заданной строки. Суффиксное дерево называется *явным*, если все суффиксы заданной строки заканчиваются в вершинах (а не на середине ребра) сжатого бора.

Будем хранить суффиксное дерево в структуре.

```
1 struct node{
2     int l, r, par; //s[l..r-1] подстрока, написанная на ведущем в вершину
    ребре
3     map<char, int> next;
4     node(int l, int r, int par) : l(l), r(r), par(par){
5     }
6 };
```

Позиция в суффиксном дереве — это положение мнимого указателя после прочтения некоторой строки, если начинать от корня.

Будем хранить позиция в дереве в следующей структуре.

```
1 struct position{
2     int v, L;
3     position(int v, int L) : v(v), L(L) {}
4 };
```

Операции с суффиксным деревом

Дополнительно определим следующие операции над суффиксным деревом.

```
1 int leng(int v); // длина ребра.
1 void split_edge(position pos); // режет ребро по позиции.
```

```
1 void add_edge_to_parent(int l, int r, int parent); // добавляет ребро к
    предку.
```

```
1 position read_char(position pos, char c); // читает символ из текущей
    позиции.
```

Реализация этих операций:

```
1 int leng(int v){
2     return t[v].r - t[v].l;
3 }
```

```
1 int add_edge_to_parent(int l, int r, int parent){
2     int nidx = szt++;
3     t[nidx] = node(l, r, parent);
4     return (t[parent].next[s[l]] = nidx);
5 }
```

```
1 int split_edge(position pos){
2     int v = pos.V, up = pos.L, down = leng(v) - up;
3
4     if (up == 0) return v;
5     if (down == 0) return t[v].par; //этот if не нужен почти всегда
6
7     int mid = add_edge_to_parent(t[v].l, t[v].l + down, t[v].par);
8     t[v].l += down, t[v].par = mid;
9     t[mid].next[s[t[v].l]] = v;
10    return mid;
11 }
```

```
1 pt read_char(position pos, char c){
2     int v = pos.V, up = pos.L;
3     if (up > 0)
4         return s[t[v].r] == c ? position(v, up - 1) : position(-1, -1);
5     else{
6         int nextv = t[v].next.count(c) ? t[v].next[c] : -1;
7         return nextv != -1 ? position(nextv, leng(nextv) - 1) : position(-1,
            -1);
8     }
9 }
```

Построение суффиксного дерева наивным алгоритмом

Пусть у нас есть строка s , пользуясь этими операциями научимся строить неявное суффиксное дерево t за время $O(|s|^2)$. Квадратичный алгоритм просто последовательно добавляет суффиксы в дерево, начиная от самого большого. Его код описан ниже.

```
1 void make_tree(){
2     node root(-1, -1, -1);
3     t[szt++] = root;
4
5     for(int i = 0; i < szs; ++i){
6         position pos(0, 0);
```

```

7         for(int j = i; j < szs; j++){
8             position npos = read_char(pos, s[j]);
9             if (npos.V != -1)
10                pos = npos;
11             else {
12                 int mid = split_edge(pos);
13
14                 add_edge_to_parent(j, szs, mid);
15
16                 break;
17             }
18         }
19     }
20 }

```

Хорошо известно, что такой алгоритм очень быстро работает на случайных тестах. Например, если сгенерировать случайную строку длины 1000000, то на любом современном компьютере описанный алгоритм будет работать менее секунды.

Свойства суффиксного дерева

Рассмотрим три важных свойства неявного суффиксного дерева.

Скажем, что две вершины (v, u) связаны суффиксной связью, если путь в вершину u является наибольшим собственным суффиксом пути в вершину v . Если вершины (v, u) связаны суффиксной связью, введем обозначение $link(v) = u$. Скажем, что количество вершин на пути от вершины v до корня дерева равно $dep(v)$.

1. В неявном суффиксном дереве для каждой строки (позиции) в дереве есть все ее суффиксы. Это утверждение очевидно следует из определения суффиксного дерева.
2. В неявном суффиксном дереве все внутренние вершины связаны суффиксной связью с некоторыми внутренними вершинами. Рассмотрим некоторую внутреннюю вершину дерева, путь до которой содержит L ($L > 1$) символов. Тогда, так как вершина — внутренняя, найдутся два суффикса x ($|x| > L$) и y ($|y| > L$) таких, что их наибольший общий префикс имеет длину ровно L . Это означает, что для вершины $link(v)$ существуют два суффикса x_1 ($|x_1| = |x| - 1$), y_1 ($|y_1| = |y| - 1$) таких, что их наибольший общий префикс имеет длину ровно $L - 1$ и ведет в вершину $link(v)$, если прочитать его от корня. Это означает, что в вершине $link(v)$, также есть развилка. Значит вершина $link(v)$ — внутренняя.
3. В неявном суффиксном дереве для любой внутренней вершины v выполняется равенство $dep(link(v)) \geq dep(v) - 1$. Рассмотрим вершины

на пути от v до корня кроме корня. Обозначим последовательность таких вершин как $x_1, x_2, \dots, x_{\text{dep}(v)}$ ($x_{\text{dep}(v)} = v$). Тогда все вершины $\text{link}(x_1), \text{link}(x_2), \dots, \text{link}(x_{\text{dep}(v)})$ будет лежать на одном пути, начинающемся в корне. Более того, все эти вершины будет различны. Это доказывает наше утверждение.

Как посчитать суффиксную ссылку в уже построенном суффиксном дереве? Для этого нужно посчитать суффиксную ссылку w для родителя вершины, а далее пройти от w по требуемой строке. Так как мы знаем, что что нужная вершина (назовем ее k) существует в дереве (по свойству 2), можно искать вершину k за $\text{dep}(k) - \text{dep}(w)$. Опишем функции для поиска суффиксной ссылки.

```

1 position fast_go_down(int v, int l, int r){
2     if (l == r) return position(v, 0);
3     while(true){
4         v = t[v].next[s[l]];
5         if (leng(v) >= r - l)
6             return position(v, leng(v) - r + l);
7         l += leng(v);
8     }
9     throw;
10 }

1 int link(int v){
2     if (t[v].link == -1)
3         t[v].link = split_edge(fast_go_down(link(t[v].par), t[v].l + int(t[v].par == 0), t[v].r)); //нельзя забывать про корень.
4     return t[v].link;
5 }

```

Можно заметить, что функция `link` вызывает `split_edge`. У уже построенном суффиксном дереве такая функция не понадобилась бы. Нам такой вызов понадобится в дальнейшем.

Алгоритм Укконена

Алгоритм Укконена строит суффиксное дерево, последовательно добавляя по символу в конец строки. Если обозначить строку $s = s_0s_1s_2 \dots s_{|s|-1}$, то алгоритм Укконена строит последовательность неявных суффиксных деревьев для строк $s_0, s_0s_1, \dots, s_0s_1 \dots s_{|s|-1}$. Переход от одного неявного суффиксного дерева к следующему будем называть добавлением следующего символа в дерево.

Алгоритм хранит указатель (позиция в дереве), на наидлиннейший суффикс, который встречается в строке ровно один раз. После добавления следующего символа в дерево этот указатель модифицируется.

Функции добавления символа и построения дерева описаны ниже:

```

1 position add_char_to_tree(position pos, int i){
2     while(true){
3         position npos = read_char(pos, s[i]);
4         if (npos.V != -1) return npos;
5
6         int mid = split_edge(pos);
7
8         add_edge_to_parent(i, szs, mid);
9
10        pos = position(link(mid), 0);
11
12        if (mid == 0)
13            return pos;
14    }
15    throw;
16 }

1 void make_tree(){
2     node root(-1, -1, -1); root.link = 0;
3     t[szt++] = root;
4
5     position pos(0, 0);
6     for(int i = 0; i < szs; ++i){
7         pos = add_char_to_tree(pos, i);
8     }
9 }

```

Задачи и разборы

Задача А. Черно-белый куб

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Вам дан куб размера $k \times k \times k$, состоящий из единичных кубиков. Два единичных кубика считаются соседними, если у них есть общая грань.

Ваша задача — покрасить каждый из k^3 единичных кубиков в один из двух цветов (черный и белый), так чтобы выполнялись два условия:

- у каждого белого кубика есть ровно 2 соседних кубика белого цвета;
- у каждого черного кубика есть ровно 2 соседних кубика черного цвета.

Формат входного файла

В первой строке входных данных задано целое число k ($1 \leq k \leq 100$) — размер куба.

Формат выходного файла

Если решения не существует, то выведите -1. Иначе выведите искомую раскраску куба последовательно по слоям. В первых k строках выведите матрицу $k \times k$ — как должен быть раскрашен первый слой куба. В следующих k строках выведите матрицу $k \times k$ — как должен быть раскрашен второй слой куба. И так далее до последнего k -го слоя. Обратите внимание, что ориентация куба в пространстве не имеет значения.

Единичный куб белого цвета обозначайте символом «w», черного — «b». Следуйте формату выходных данных, который указан в тестовых примерах. При проверке правильности ответа пустые строки никак не учитываются.

Примеры

stdin	stdout
1	-1
2	bb ww bb ww

Разбор задачи А. Черно-белый куб

1. Рассмотрим граф, в котором вершины - это единичные кубики, а между вершинами есть ребро тогда и только тогда, когда соответствующие кубики имеют общую грань.
2. В этом графе не существует циклов нечетного веса.
3. Рассмотрим раскраску этого графа, соответствующую условию задачи.
4. Эта раскраска соответствует разбиению графа на циклы, так как у каждого кубика одного цвета есть два соседа такого же цвета, как и вершин в цикле есть ровно два соседа. Так как все циклы имеют четную длину, то количество вершин в графе четно. Следовательно для нечетных k ответа не существует.
5. Рассмотрим задачу на плоскости. Для квадрата с четной стороной можно построить ответ следующего вида:

```

wwwwwwww
wbbbbbbw
wbwwwbw
wbwbbwbw
wbwbbwbw
wbwwwbw
wbbbbbbw
wwwwwwww

```

6. Пусть мы знаем ответ для квадрата $n \times n$, то мы можем получить ответ для куба $n \times n \times n$.
7. Четный слой куба будет равен ответу для квадрата, а нечетный - его инверсии (т.е. черные кубики будут белыми, а белые - черными). Таким образом для каждого единичного кубика не изменится количество соседних кубиков такого же цвета.

Задача В. Один-два

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

У Коня Валеры есть целое число n . Также у Коня Валеры есть копыта. Копытами писать сложно, поэтому Валера умеет писать только цифры один («1») и два («2»).

Сейчас Валера хочет найти целое положительное число, которое будет делиться на 2^n без остатка и при этом будет содержать ровно n цифр в десятичной записи. Конечно, десятичная запись этого числа не должна содержать никаких цифр кроме единиц и двоек.

Помогите Валере, найдите и выведите искомое число.

Формат входного файла

В первой строке задано целое число n ($1 \leq n \leq 50$) — число Валеры.

Формат выходного файла

Выведите целое положительное число в десятичной системе счисления, состоящее из n цифр, которое будет делиться на 2^n . Десятичная запись этого числа может содержать только единицы и двойки.

Гарантируется, что ответ существует. Если существует несколько ответов, разрешается вывести любой.

Примеры

stdin	stdout
1	2
2	12

Разбор задачи В. Один-два

1. Докажем, что ответов не более одного. Сделаем это по индукции. Для $n = 1$ ответ единственный.
2. Пусть для n ответ единственный и равен a .
3. Рассмотрим ответ для $n + 1$. Последние n цифр ответа для $n + 1$ являются ответом для n . Так как ответ для n единственный, то существует максимум два различных ответа для $n + 1$: $10^n + a$ и $2 \cdot 10^n + a$. Обозначим их как b и c соответственно.
4. Так как b и c - ответы для $n + 1$, то $b \bmod 2^{n+1} = c \bmod 2^{n+1} = 0$. Значит $(b - c) \bmod 2^{n+1} = 0$. Но $b - c = 10^n = 2^n \cdot 5^n$. Так как это число не делится на 2^{n+1} , получили противоречие, а значит ответов не более одного.
5. Так как ответов не более одного для каждого n , то задачу можно решать итеративно. Для $n = 1$ ответ известен. Пусть известен ответ для n . Тогда переберем какую цифру мы добавим к ответу для n и проверим, что полученное число — ответ для $n + 1$.

Задача С. Оптимальное разрезание

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

У Поликарпа есть прямоугольный кусочек клетчатой бумаги $n \times m$ клеток. В каждой клетке бумаги записана буква.

Две клетки назовем *связными*, если они имеют общую сторону и в них записана одна и та же буква.

Одноцветной областью назовем множество клеток бумаги таких, что любые две клетки этого множества связаны напрямую или косвенно (через последовательность других клеток) и при этом нельзя увеличить это множество, не нарушая предыдущего свойства.

Сторона клетки называется *внутренней*, если она отделяет друг от друга две какие-то клетки бумаги.

Путем назовем последовательность различных внутренних сторон клеток бумаги такую, что любые две последовательные стороны пересекаются в углу какой-то клетки.

Поликарп хочет отсоединить все одноцветные области друг от друга, разрезая бумагу. Одним разрезом разрешается разрезать последовательность **ранее не разрезанных** сторон, образующую некоторый путь. Какое минимальное количество разрезов ему для этого понадобится? Учтите, что после применения разрезов одноцветные области не должны оказаться разрезанными.

Формат входного файла

В первой строке записано два целых положительных числа через пробел n и m ($1 \leq n \times m \leq 10^5$) — размеры кусочка бумаги.

В следующих n строках записано по m строчных букв латинского алфавита — описание кусочка бумаги.

Формат выходного файла

Выведите единственное целое число — минимальное количество разрезов, требуемое для отделения цветных областей друг от друга.

Примеры

stdin	stdout
2 2 ab ba	2
1 9 aaabbbccc	2
4 4 xyzx yzxx zxxy xxyz	5
1 1 a	0

Разбор задачи С. Оптимальное разрезание

1. Заметим, что если какая-либо сторона разделяет две клетки с разными буквами, то мы обязаны её разрезать, а если с одинаковыми, то мы обязаны её не разрезать.
2. Построим граф, в котором вершинами являются углы поля, а ребрами - стороны, которые мы обязаны разрезать. Этот граф необходимо разбить на минимальное количество простых, но возможно циклических, путей, так, чтобы каждое ребро принадлежало ровно одному простому пути.
3. Для каждой компоненты связности задачу можно решать независимо, а потом сложить ответ для всех компонент.
4. Если в графе нет нечетных вершин, или нечетных вершин ровно две, то в графе существует Эйлеров цикл или путь, а значит ответ равен единице.
5. Если удалить из графа один путь, то количество нечетных вершин уменьшится максимум на два. А значит, чтобы в графе, в котором x нечетных вершин необходимо провести $(x - 2)/2$ путей, чтобы в графе появился Эйлеров путь.
6. Значит ответ для графа с $x \leq 2$ нечетными вершинами равен $(x - 2)/2 + 1$, а если в графе нет нечетных вершин, то ответ 1, так как существует Эйлеров цикл.

Задача D. Граф-турнир

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

В данной задаче от Вас требуется построить граф-турнир, состоящий из n вершин, такой, что для любой ориентированной пары вершин $(v, u) (v \neq u)$ существует путь из вершины v в вершину u длины не более двух ребер.

Ориентированный граф без петель называется *турниром*, если между любыми его двумя различными вершинами есть ровно одно ребро (в одном из двух возможных направлений).

Формат входного файла

В первой строке записано единственное целое число n ($3 \leq n \leq 1000$) — количество вершин графа.

Формат выходного файла

Выведите -1 , если не существует ни одного графа, удовлетворяющего описанным условиям.

Иначе выведите n строк, в каждой строке по n чисел, разделенных пробелами, — матрицу смежности a найденного турнира. Считайте, что вершины графа пронумерованы целыми числами от 1 до n . Тогда $a_{v,u} = 0$, если в турнире нет ребра из вершины v в вершину u , и $a_{v,u} = 1$, если ребро есть.

Так как выведенный граф должен быть турниром, то должны выполняться следующие равенства:

- $a_{v,u} + a_{u,v} = 1$ для всех v, u ($1 \leq v, u \leq n; v \neq u$);
- $a_{v,v} = 0$ для всех v ($1 \leq v \leq n$).

Примеры

stdin	stdout
3	0 1 0 0 0 1 1 0 0
4	-1

Разбор задачи D. Граф-турнир

1. Пусть мы нашли ответ для $n \geq 3$. Тогда мы можем получить ответ для $n + 2$ следующим образом.
2. Добавим две новые вершины: s, t .
3. Добавим ребро из t в s .
4. Для любой старой вершины v добавим два ребра: из s в v и из v в t .
5. Таким образом расстояние от s до любой старой вершины равно 1. Расстояние от s до t равно 2.

6. Расстояние от любой старой вершины до t равно 1. Расстояние от любой старой вершины до s равно 2.
7. Расстояние от t до s равно 1. Расстояние от t до любой старой вершины равно 2.
8. Так как нам известен ответ для $n = 3$, то мы можем получить ответ для любого нечетного $n \geq 3$, повторив операцию добавления вершины необходимое число раз.
9. Для $n = 4$, как показывает перебор всех возможных графов, ответа не существует.
10. Ответ существует для $n = 6$. Если найти его, например, перебором, то можно получить ответ для любого четного $n \geq 6$.

Задача Е. Замок в виде мини-игры

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Во многих компьютерных играх взлом замков реализуется в виде мини-игр. В своей игре Поликарп хочет использовать следующую мини-игру.

На экране написаны три целых неотрицательных числа: a, b, c . Также есть три кнопки «а», «b», «с».

- Кнопка «а»: прибавляет к числу a единицу, а из чисел b и c вычитает единицу. Эту кнопку можно нажать, только если числа b и c строго больше нуля.
- Кнопка «b»: прибавляет к числу b единицу, а из чисел a и c вычитает единицу. Эту кнопку можно нажать, только если числа a и c строго больше нуля.
- Кнопка «с»: прибавляет к числу c единицу, а из чисел a и b вычитает единицу. Эту кнопку можно нажать, только если числа a и b строго больше нуля.

Замок считается открытым, если после некоторой (возможно, пустой) последовательности нажатий на кнопки сумма чисел a, b и c равна единице.

Ваша задача состоит в том, чтобы по числам на экране сказать, какое минимальное количество нажатий потребуется, чтобы открыть замок, или

определить, что это невозможно. Если замок можно открыть, требуется вывести любой оптимальный способ сделать это.

Формат входного файла

В первой строке заданы три целых неотрицательных числа: a, b, c ($0 \leq a, b, c \leq 10^9, 1 \leq a + b + c \leq 10^9$) — числа, которые изначально написаны на экране.

Формат выходного файла

В первой строке выведите единственное целое число — минимальное количество нажатий, требуемое для открытия замка. Если открыть замок невозможно, выведите -1 .

Если замок открыть можно, во второй строке выведите любой оптимальный способ сделать это — строку, состоящую из символов «a», «b», «c», « $\left[\right]$ », « $\left[\right]$ » и цифр. Строка должна содержать не более 1000 символов. Строка должна удовлетворять грамматике с начальным символом `answer`:

```
character ::= a | b | c;
sequence ::= character | character sequence;
block ::= character | number [ sequence ];
answer ::= block | block answer | <<пустая строка>>;
```

Здесь `number` ($1 \leq \text{number} < 10^{10}$) — целое положительное число, записанное без лидирующих нулей. Обратите внимание, что строка, удовлетворяющая грамматике, не содержит пробелов.

Выведенная строка должна обозначать последовательность нажатий в оптимальном ответе. Блок `character` соответствует нажатию на соответствующую кнопку, блок `number [sequence]` соответствует повторению `number` раз последовательности нажатий `sequence`. Смотрите тестовые примеры для лучшего понимания.

Примеры

stdin	stdout
1 1 0	1 c
2 0 0	-1
0 0 1	0
5 4 5	13 4[abc]b

Разбор задачи Е. Замок в виде мини-игры

1. Все конфигурации чисел на экране рассматриваются с точностью до перестановки.
2. Замок открыт, когда числа на экране имеют вид 001.
3. Если все числа имеют одинаковую четность, то ответа не существует, так как после нажатия на любую кнопку все числа изменят четность (т.е. все числа опять будут одинаковой четности), а в ответе есть числа с различной четностью.
4. Значит будем считать, что существуют два числа с различной четностью.
5. Заметим, что за одно нажатие сумма чисел уменьшается ровно на 1, а значит, если ответ существует, то для открытия замка потребуется нажатий ровно сумма всех чисел минус один.
6. Если начальная конфигурация имеет вид 001, то ответ — ничего не нажимать, замок и так открыт.
7. Если начальная конфигурация имеет вид $00a(a \neq 1)$, то ответа не существует, так как нельзя нажать ни одну кнопку.
8. Если начальная конфигурация имеет вид $01a$, то ответ существует, так как за одно нажатие мы можем перейти в конфигурацию $10a - 1$, а значит в $01a - 1$. Таким образом за a операций мы перейдем в состояние 010.
9. Если конфигурация имеет вид $aa + 1b$, то ответ существует, так как за a операций мы можем перейти в конфигурацию $01b + a$.
10. Пусть мы находимся в конфигурации abc . За две операции мы можем перейти в конфигурацию $a - 2bc$ следующим образом: $a - 1b - 1c + 1$, $a - 1 - 1b - 1 + 1c + 1 - 1$, то есть $a - 2bc$.
11. Рассмотрим два числа разной четностью. Пусть меньшее из них равно a , а больше равно $a + b$. Так как числа разной четности, то число b - нечетное.
12. Значит, если мы будем уменьшать число $a + b$ на 2, то через $\frac{b-1}{2}$ операций мы придем в конфигурацию $aa + 1c$, а значит найдем ответ.

Задача F. Количество запросов

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Даны два целых числа n, k . Рассмотрим функцию обработки запроса $[from, to]$ деревом отрезков из вершины idx , которая соответствует отрезку массива $[left, right]$ ($from \leq to, left \leq right$).

```
1 int get(int idx, int left, int right, int from, int to) {
2     int ans = 1;
3
4     if (left == from && right == to)
5         return ans;
6     int mid = (left + right) / 2;
7
8     if (from <= mid)
9         ans += get(2 * idx + 1, left, mid, from, min(mid, to));
10    if (to >= mid + 1)
11        ans += get(2 * idx + 2, mid + 1, right, max(from, mid + 1), to);
12
13    return ans;
14 }
```

Обратите внимание, что эта функция может отличаться от того как Вы пишете дерево отрезков. Пожалуйста, прочитайте ее внимательно.

Нужно найти количество запросов $[from, to]$ ($0 \leq from \leq to \leq n - 1$), таких что функция $get(0, 0, n - 1, from, to)$ вернет в качестве ответа k . Выведите остаток от деления ответа на задачу на 1000000009 ($10^9 + 9$).

Формат входного файла

В первой строке записаны через пробел два целых числа: n ($1 \leq n \leq 10^{18}$), k ($0 \leq k \leq 10^{18}$).

Формат выходного файла Выведите одно целое число — ответ на задачу по модулю 1000000009 ($10^9 + 9$).

Примеры

stdin	stdout
6 5	4
10 6	9
4 6	0

Разбор задачи F. Количество запросов

1. Заметим, что функция $\text{get}(\text{idx}, l, r, \text{from}, \text{to})$ возвращает количество вершин дерева отрезков, которые были посещены при обработке запроса.
2. Так как дерево отрезков обрабатывает один запрос за $O(\log n)$, то если k больше $10 \cdot \log n$, то ответ точно 0.
3. Задачу можно решать динамическим программированием вида $z[L][V][K]$, где V — текущая вершина дерева отрезков, K — количество вершин, которые необходимо посетить в этом поддереве, а L — параметр, определяющий множество допустимых отрезков.
4. Пусть вершина V соответствует отрезку массива от l до r .
5. Тогда если $L = 0$, то можно рассматривать любой отрезок $[\text{from}, \text{to}]$, такой что $l \leq \text{from} \leq \text{to} \leq r$.
6. Если $L = 1$, то можно рассматривать любой отрезок $[l, \text{to}]$, такой что $l \leq \text{to} \leq r$.
7. Если $L = 2$, то можно рассматривать любой отрезок $[\text{from}, r]$, такой что $l \leq \text{from} \leq r$.
8. Для вычисления динамики с параметрами (L, V, K) необходимо:
9. Если $k < 1$, то ответ — 0, так как минимум вершина V уже посещена.
10. Если $k = 1$, то единственным способом решить задачу будет взять отрезок $[l, r]$. Таким образом будет посещена лишь вершина V .
11. Рассмотрим $\text{mid} = \frac{l+r}{2}$.
12. Если $L = 1$, то существует два способа взять отрезок $[l, \text{to}]$.
13. Если $\text{to} \leq \text{mid}$, то вычисления в дереве отрезков будут происходить только в левом поддереве, то есть к ответу необходимо прибавить $z[1][2 \cdot V + 1][K - 1]$.
14. Если $\text{mid} < \text{to}$, то вычисления в дереве отрезков будут происходить только в правом поддереве, а левое поддерево будет посещено только вершина $2 \cdot V + 1$, то есть к ответу необходимо прибавить $z[1][2 \cdot V + 2][K - 2]$.

15. Для случая $L = 2$ аналогично.
16. Если $L = 0$, то существует три способа взять отрезок $[from, to]$.
17. Если $to \leq mid$, то вычисления в дереве отрезков только в левом поддереве, то есть к ответу необходимо прибавить $z[0][2 \cdot V + 1][K - 1]$.
18. Если $mid < from$, то вычисления в дереве отрезков только в правом поддереве, то есть к ответу необходимо прибавить $z[0][2 \cdot V + 2][K - 1]$.
19. Иначе, вычисления будут происходить как слева, так и справа. Необходимо перебрать параметр lk — количество вершин, которое необходимо посетить в левом поддереве и для всех значений lk от 1 до $k - 2$ прибавить к ответу $z[2][2 \cdot V + 1][lk] \times z[1][2 \cdot V + 2][k - 1 - lk]$.
20. Заметим, что при вычислениях не используется информация о том, что мы стоим в какой-то конкретной вершине дерева отрезком. Важна лишь длина отрезка массива, соответствующая этой вершине.
21. Всего различных длин отрезков в дереве $O(\log n)$. Доказательство.
22. В дереве отрезков $O(\log n)$ уровней. На первом уровне существует ровно одна длина n .
23. Пусть на уровне i одна длина x . Тогда если x четное, то на уровне $i + 1$ одна длина $\frac{x}{2}$, иначе две длины $\frac{x-1}{2}$ и $\frac{x-1}{2} + 1$.
24. Пусть на уровне i две длины: x и $x + 1$. Тогда если x четное, то на $i + 1$ будет две длины $\frac{x}{2}$ и $\frac{x}{2} + 1$, иначе тоже две длины: $\frac{x+1}{2}$ и $\frac{x+1}{2} - 1$.
25. Таким образом на каждом из $O(\log n)$ уровней не более двух длин, а значит их тоже $O(\log n)$.
26. Таким образом в динамике $O(k \cdot \log n)$ состояний и из некоторых из них есть переход за $O(k)$. То есть время работы $O(k^2 \cdot \log n)$.
27. Так как $k \leq 10 \cdot \log n$, то итоговая асимптотика решения $O(\log^3 n)$.
28. На самом деле асимптотика этого решения $O(\log^2 n)$. Над доказательством этого факта вам предлагается подумать самому.

Задача G. Две перестановки

Вход:	stdin
Выход:	stdout
Ограничение по времени:	5 с
Ограничение по памяти:	512 Мб

Вам даны две перестановки p и q , состоящие из n элементов, и m запросов вида: l_1, r_1, l_2, r_2 ($l_1 \leq r_1$; $l_2 \leq r_2$). Ответом на запрос является количество целых чисел от 1 до n , таких что их позиция в первой перестановке находится в отрезке $[l_1, r_1]$ (границы отрезка включаются), а позиция во второй перестановке — в отрезке $[l_2, r_2]$ (границы отрезка включаются).

Перестановкой, состоящей из n элементов, называется последовательность n различных целых чисел, каждое из которых не меньше 1 и не больше n .

Позицией числа v ($1 \leq v \leq n$) в перестановке g_1, g_2, \dots, g_n называется такое число i , что $g_i = v$.

Формат входного файла

В первой строке находится одно целое число n ($1 \leq n \leq 10^6$) — количество элементов в обеих перестановках. В следующей строке находятся n чисел, разделенных пробелами: p_1, p_2, \dots, p_n ($1 \leq p_i \leq n$) — элементы первой перестановки. В следующей строке находится вторая перестановка q_1, q_2, \dots, q_n в таком же формате.

В следующей строке находится одно целое число m ($1 \leq m \leq 2 \cdot 10^5$) — количество запросов.

В следующих m строках находятся описания запросов по одному в строке. Описание i -того запроса состоит из четырех целых чисел: a, b, c, d ($1 \leq a, b, c, d \leq n$). Параметры запроса l_1, r_1, l_2, r_2 получаются из чисел a, b, c, d следующим образом:

1. Введем переменную x . Если запрос первый, то она равна 0, иначе она равна ответу на предыдущий запрос плюс один.
2. Введем функцию $f(z) = ((z - 1 + x) \bmod n) + 1$.
3. Положим $l_1 = \min(f(a), f(b))$, $r_1 = \max(f(a), f(b))$, $l_2 = \min(f(c), f(d))$, $r_2 = \max(f(c), f(d))$.

Формат выходного файла

Для каждого запроса выведите одно целое число в отдельной строке — ответ на запрос.

Примеры

stdin	stdout
3 3 1 2 3 2 1 1 1 2 3 3	1
4 4 3 2 1 2 3 4 1 3 1 2 3 4 1 3 2 1 1 4 2 3	1 1 2

Разбор задачи G. Две перестановки

1. Не нарушая общности можно считать, что первая перестановка тождественная. Вторую перестановку будем обозначать p_1, p_2, \dots, p_n .
2. Тогда ответ на запрос (l_1, r_1, l_2, r_2) — это количество чисел таких чисел x , что $l_1 \leq x \leq r_1$ и позиция x в заданной перестановке находится в отрезке $[l_2, r_2]$.
3. Рассмотрим другой тип запросов (l_1, r_1, R) — это количество чисел таких чисел x , что $l_1 \leq x \leq r_1$ и позиция x в заданной перестановке находится в отрезке $[1, R]$.
4. Тогда запрос (l_1, r_1, l_2, r_2) можно представить в виде $(l_1, r_1, r_2) - (l_1, r_1, l_2 - 1)$.
5. Предположим, что параметры всех запросов известны заранее. Тогда задачу можно решать следующим образом.
6. Построим дерево отрезков над числами от 1 до n , которое умеет считать сумму на отрезке. Изначально оно заполнено нулями.
7. Будем идти по перестановке слева направо. В начале итерации i увеличим число в позиции p_i на единицу. Теперь мы можем посчитать ответ на любой запрос вида (l_1, r_1, i) — это будет сумма на отрезке $[l_1, r_1]$ в текущем дереве отрезков.

8. Так как запросы заранее не известны, то необходимо использовать персистентное дерево отрезком, чтобы иметь доступ ко всем версиям дерева отрезком.

Задача Н. Робот-конь

Вход:	stdin
Выход:	stdout
Ограничение по времени:	5 с
Ограничение по памяти:	256 Мб

В стране коней живет робот-конь. У робота коня есть матрица размером $n \times m$. В каждой ячейке матрицы записан вектор. На пересечении строки i и столбца j записан вектор $(x_{i,j}, y_{i,j})$ ($1 \leq i \leq n, 1 \leq j \leq m$).

Робот-конь решил прогуляться, для этого он составил себе программу, состоящую из k шагов. На шаге номер t ($1 \leq t \leq k$) робот-конь может подвинуться из своей текущей позиции на какой-то вектор из строки матрицы с номером $((t-1) \bmod n) + 1$. Также на любом шаге можно никуда не двигаться, то есть остаться на месте.

Вам задана матрица, которая есть у робота-коня, и число k . На какое максимальное расстояние робот-конь сможет удалиться от своей начальной позиции? Найдите квадрат этого расстояния.

Формат входного файла

В первой строке записаны три целых числа n, m, k ($1 \leq n, m \leq 10^5, 1 \leq k \leq 10^9, n \times m \leq 10^5$) — размеры матрицы и количество шагов робота-коня.

В каждой из следующих n строк записаны по $2 \cdot m$ целых чисел: в строке с номером i записаны числа $x_{i,1}, y_{i,1}, x_{i,2}, y_{i,2}, \dots, x_{i,m}, y_{i,m}$ ($|x_{i,j}|, |y_{i,j}| \leq 10^9$).

Формат выходного файла

Выведите единственное целое число — квадрат максимального расстояния, на которое сможет удалиться от своей начальной позиции робот-конь.

Примеры

stdin	stdout
2 2 3 1 0 -1 0 0 1 0 -1	5
2 1 10 1 1 -1 -1	50
2 3 1000000000 1001320 -1000 123 1233 2233 -1232 -12300 -10 -1233 -15533 2233 -123	25177997123450000000000000000000

Разбор задачи Н. Робот-конь

1. Очевидно, что для каждой строки матрицы необходимо всегда выбирать один и тот же вектор или всегда никуда не двигаться. Таким образом необходимо умножить все вектора строки i на то, сколько раз робот будет делать ход из этой строки.
2. Теперь задачу можно сформулировать следующим образом. Есть n наборов векторов по $m + 1$ вектором в каждом. В каждом наборе есть вектор $(0, 0)$. Необходимо из каждого набора выбрать ровно один вектор так, чтобы длина суммы выбранных векторов была максимальна.
3. Для каждого набора векторов построим выпуклую оболочку. Найдем сумму Минковского полученных оболочек.
4. Очевидно, что вершины полученного многоугольника соответствуют тому, что мы взяли из каждого набора по вектору, а значит для вычисления ответа необходимо перебрать только лишь эти вершины.

Задача I. Прыг-скок

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Страна коней — это шахматная доска размера $1 \times n$, клетки которой пронумерованы от 1 до n слева направо. В некоторых клетках доски стоят кони, не более одного коня в одной клетке. Первый конь стоит в клетке с номером a_1 , второй стоит в клетке с номером a_2 , последний конь стоит в клетке с номером a_k — всего k коней.

Король коней и его брат играют в игру на описанной шахматной доске. Игроки ходят по очереди, король коней ходит первым. За один ход игрок может выбрать любого коня на шахматной доске и переставить его в ближайшую свободную клетку справа от выбранного коня. Клетка считается свободной, если в ней нет коней. Выигрывает тот игрок, после хода которого хотя бы один конь окажется в клетке с номером n .

Так как король коней ходит первым, он хочет знать сколько у него существует выигрышных ходов. Другими словами, сколько существует таких ходов, сделав которые, можно выиграть независимо от действий противника. Считайте, что оба игрока играют оптимально.

Формат входного файла

В первой строке входных данных записаны целые числа n и k ($1 \leq n \leq 10^9, 1 \leq k \leq 10^6$) — количество клеток доски и количество коней на ней.

Во второй строке записана строго возрастающая последовательность a_1, a_2, \dots, a_k ($1 \leq a_i < n, a_i < a_{i+1}$) — позиции коней на доске.

Гарантируется, что в клетке номер n не стоит ни один конь.

Формат выходного файла

В единственную строку выведите единственное целое число — количество выигрышных ходов из заданной позиции.

Примеры

<code>stdin</code>	<code>stdout</code>
5 2 1 3	1
5 2 2 3	0

Разбор задачи I. Прыг-скок

1. Если изначально в клетке $n - 1$ находится хотя бы один конь, то ответ — это количество подряд идущих коней, начиная с этого коня.
2. Иначе проиграет тот игрок, который поставит коня в клетку $n - 1$, так как следующим ходом другой игрок выиграет.
3. Тогда можно удалить клетки n и $n - 1$ и сказать, что проигрывает игрок, который не может сделать ход.
4. Для каждого коня посчитаем количество свободных клеток, справа от него.
5. Рассмотрим таблицу юнга, где столбцам соответствуют кони, а высота столбца i — это количество свободных клеток, справа от i -того коня.
6. Рассмотрим процесс хода. Во время хода каким-либо конем количество свободных клеток справа от коня уменьшается на 1. Так же оно уменьшается на 1 для всех коней, стоящих подряд за i -ым.
7. Если повернуть представленную таблицу юнга и посмотреть на неё как на **ним**, то можно заметить, что ход — это взятие какого-либо количества камней из одной из кучек, при том, что нельзя оставлять камней меньше, чем в следующей кучке.
8. Мы свели задачу к нему, где если есть набор кучек a_1, a_2, \dots, a_m ($a_i \leq a_{i+1}$). Ход — это взятие из любой кучки любого количества камней таким образом, что неравенство $a_i \leq a_{i+1}$ не нарушается.
9. Рассмотрим последовательность d_1, d_2, \dots, d_m , где $d_1 == a_1, d_i = a_i - a_{i-1}$. Рассмотрим ним на этой последовательности. Этот ним состоит в том, что если мы забираем x камней из одной кучки, то мы добавляем x в соседнюю кучку.
10. Эта известная разновидность нима. Называется лестничный ним. Найти решение для этого нима можно в Интернете.

Задача J. Суффиксное дерево

Вход: `stdin`
Выход: `stdout`
Ограничение по времени: 1 с
Ограничение по памяти: 256 Мб

Дана строка s . Постройте сжатое суффиксное дерево для строки s и выведите его. Найдите такое дерево, которое содержит минимальное количество вершин.

Формат входного файла

В первой строке записана строка s ($1 \leq |s| \leq 10^5$), последний символ строки доллар «\$», остальные символы строки маленькие латинские буквы.

Формат выходного файла

Пронумеруйте вершины дерева от 0 до $n - 1$ в порядке обхода в глубину, обходя поддеревья в порядке лексикографической сортировки исходящих из вершины ребер. Используйте ASCII-коды символов для определения их порядка.

В первой строке выведите целое число n — количество вершин дерева. В следующих $n - 1$ строках выведите описание вершин дерева, кроме корня, в порядке увеличения их номеров.

Описание вершины дерева v состоит из трех целых чисел: p, lf, rg , где p ($0 \leq p < n, p \neq v$) — номер родителя текущей вершины. На ребре, ведущем из p в v , написана подстрока $s[lf \dots rg - 1]$ ($0 \leq lf < rg \leq |s|$).

Примеры

stdin	stdout
aaa\$	7 0 3 4 0 0 1 2 3 4 2 1 2 4 3 4 4 2 4
b\$	3 0 1 2 0 0 2
ababa\$	10 0 5 6 0 0 1 2 5 6 2 1 3 4 5 6 4 3 6 0 1 3 7 5 6 7 3 6

Примечание

Подстрокой $s[l...r]$ ($0 \leq l \leq r < |s|$) строки $s = s_0s_1 \dots s_{|s|-1}$ (где $|s|$ — длина строки s) называется строка $s_ls_{l+1} \dots s_r$.

Разбор задачи J. Суффиксное дерево

В этой задаче необходимо реализовать то, что было рассказано на лекции.

Задача K. Суффиксное дерево двух строк

Вход: stdin
Выход: stdout
Ограничение по времени: 1 с
Ограничение по памяти: 256 Мб

Даны строки s и t . Постройте сжатое суффиксное дерево, которое со-

держит все суффиксы строки s и строки t . Найдите такое дерево, которое содержит минимальное количество вершин.

Формат входного файла

В первой строке записана строка s ($1 \leq |s| \leq 10^5$), последний символ строки равен «\$», остальные символы строки маленькие латинские буквы.

Во второй строке записана строка t ($1 \leq |t| \leq 10^5$), последний символ строки равен «#», остальные символы строки маленькие латинские буквы.

Формат выходного файла

Пронумеруйте вершины дерева от 0 до $n - 1$ в порядке обхода в глубину, обходя поддеревья в порядке лексикографической сортировки исходящих из вершины ребер. Используйте ASCII-коды символов для определения их порядка.

В первой строке выведите целое число n — количество вершин дерева. В следующих $n - 1$ строках выведите описание вершин дерева, кроме корня, в порядке увеличения их номеров.

Описание вершины дерева v состоит из четырех целых чисел: p, w, lf, rg , где p ($0 \leq p < n, p \neq v$) — номер родителя текущей вершины, w ($0 \leq w \leq 1$) — номер строки для определения подстроки на ребре. Если $w = 0$, то на ребре, ведущем из p в v , написана подстрока $s[lf \dots rg - 1]$ ($0 \leq lf < rg \leq |s|$). Если $w = 1$, то на ребре, ведущем из p в v , написана подстрока $t[lf \dots rg - 1]$ ($0 \leq lf < rg \leq |t|$).

Примеры

stdin	stdout
ab\$ ac#	8 0 1 2 3 0 0 2 3 0 0 0 1 3 0 1 3 3 1 1 3 0 0 1 3 0 1 1 3
aba\$ baab#	14 0 1 4 5 0 0 3 4 0 0 0 1 3 0 3 4 3 1 2 5 3 0 1 2 6 1 4 5 6 0 2 4 0 0 1 2 9 1 4 5 9 0 2 3 11 0 3 4 11 1 2 5

Примечание

Подстрокой $s[l...r]$ ($0 \leq l \leq r < |s|$) строки $s = s_0s_1 \dots s_{|s|-1}$ (где $|s|$ — длина строки s) называется строка $s_ls_{l+1} \dots s_r$.

Разбор задачи К. Суффиксное дерево двух строк

1. Построим суффиксное дерево для строки $s + t$.
2. Оно содержит все суффиксы строки t .
3. Оно содержит все суффиксы строки s , но после каждого суффикса строки s идет строка t .
4. Чтобы обрезать лишние строки, необходимо при обходе в глубину полученного дерева не идти никуда, если было пройдено ребро, заканчивающееся на «\$».

Задача L. Уникальные суффиксы

Вход: `stdin`
Выход: `stdout`
Ограничение по времени: 1.5 с
Ограничение по памяти: 256 Мб

У Вас есть изначально пустая строка s . Далее поступает q запросов. Запросы бывают двух типов:

1. Запрос на добавление символа в конец строки s . Формат запроса имеет вид «+ c », где c — символ, который требуется дописать в конец строки s .
2. Запрос на проверку уникальности суффикса строки s . Формат запроса имеет вид «? l », где l — длина суффикса текущей строки s , который требуется проверить на уникальность. Суффикс считается *уникальным*, если он встречается в строке s в качестве подстроки ровно один раз (начиная с позиции $|s| - l + 1$, если считать символы строки один-индексированными).

Ваша задача — после каждого запроса второго типа, вывести «+», если заданный суффикс уникален, или вывести «-» в противном случае.

Формат входного файла

В первой строке записано единственное целое число q ($1 \leq q \leq 2 \cdot 10^6$) — количество запросов.

В следующих q строках записаны запросы в формате, описанном в условии задачи. Гарантируется, что все запросы корректны. Гарантируется, что первый запрос имеет первый тип. Гарантируется, что символ c в каждом запросе первого типа является одним из символов «a», «b», «c», «d», «e». Гарантируется, что число l во всех запросах второго типа положительно и не больше текущей длины строки s .

Формат выходного файла

Выведите ответы на запросы второго типа. Ответы на запросы выводите в порядке появления запросов во входных данных.

Примеры

stdin	stdout
13 + a ? 1 + a ? 1 ? 2 + a ? 1 ? 2 ? 3 + b ? 3 ? 1 ? 2	+ - + - - + + + +
8 + a + b + a + b + a ? 3 ? 2 ? 4	- - +

Разбор задачи L. Уникальные суффиксы

Будем строить суффиксное дерево для строки в online алгоритмом Укконена. В алгоритме поддерживается указатель на наидлиннейший суффикс, который встречается в строке более одного раза. Этот указатель — все что нужно для решения задачи. Нужно поддерживать его глубину в суффиксном дереве, и при поступлении запроса на уникальность просто сравнить длину суффикса с глубиной указателя.

Задача М. Общая подстрока

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Заданы две строки s , t и целое число k . Рассмотрим множество всех таких непустых строк, которые встречаются как подстроки в s и t одновременно. Найдите k -ую в лексикографическом порядке строку из этого множества.

Формат входного файла

В первых двух строках записаны строки s и t ($1 \leq |s|, |t| \leq 10^5$). В третьей строке записано целое число k ($1 \leq k \leq 10^{18}$).

Строки состоят из маленьких латинских букв.

Формат выходного файла

В первой строке выведите искомую строку или -1 , если такой не существует.

Примеры

stdin	stdout
aaa abaa 3	-1
aabbbabba abbabbab 1	a
baab bbaababb 3	aab
bbbbaaba abbabbabba 2	ab

Разбор задачи М. Суффиксное дерево двух строк

1. Построим суффиксное дерево для строки $s + \$ + t + \#$.
2. Если из вершины можно дойти по дереву до символа $\$$, то эта вершина соответствует подстроке строки s .
3. Если из вершины можно дойти по дереву до символа $\#$ не проходя через $\$$, то эта вершина соответствует подстроке строки t .
4. Оставим в дереве только вершины, которые соответствуют как подстроке строки s , так и подстроке строки t .
5. Таким образом мы получили бор, содержащий только строки, соответствующие нашему критерию.
6. k -ая в лексикографическом порядке строка в боре ищется при помощи модификации обхода в глубину.

Задача N. Контролирующее множество строк

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Введем отношение контроля на множестве строк. Строка a *контролирует* строку b , если либо a является префиксом b , либо b является префиксом a .

Вам задана непустая строка s . Найдите мультимножество **непустых подстрок** строки s , состоящее из ровно k строк, такое, что каждый суффикс строки s контролируется хотя бы одной строкой из этого множества и сумма длин строк, содержащихся в множестве, максимальна.

Все подстроки найденного множества не обязательно должны быть различны.

Формат входного файла

В первой строке записана строка s ($1 \leq |s| \leq 10^5$) — заданная строка. Во второй строке записано целое число k ($1 \leq k \leq \min(100, |s|)$) — требуемое количество подстрок в множестве.

Заданная строка состоит только из строчных латинских букв.

Формат выходного файла

Выведите единственное целое число — сумма длин строк оптимального множества. Если описанного множества не существует, выведите -1 .

Примеры

stdin	stdout
aba 1	-1
aaa 1	3
abacaba 3	11

Разбор задачи N. Суффиксное дерево двух строк

1. Построим суффиксное дерево для строки s .
2. Каждая вершина суффиксного дерева соответствует какой-либо подстроке строки s .
3. Так же у каждой вершины есть длина — это сумма длин подстрок, написанных на пути от корня до этой вершины.
4. Теперь задачу можно решить при помощи динамического программирования с параметрами $z[V][K]$, где V — это вершина суффиксного дерева, а K — это количество вершин, которые мы хотим взять в наше мультимножество из поддерев вершины V .
5. Если мы возьмем вершину V в множество, то все суффиксы, которые находятся в поддереве вершины V и на пути от корня до вершины V окажутся контролируемыми. А значит в качестве оставшиеся $k - 1$ вершин выгоднее всего взять вершины с максимальной длиной из поддерев вершины V .
6. Иначе необходимо взять как минимум по одной вершины из поддеревьев сыновей вершины V .
7. Такой переход реализуется следующим образом. Динамическое программирование $z_2[V][U][K]$, где V — это предок вершины U , а K — это количество вершин, которые мы хотим взять в наше мультимножество из поддерев вершины V .

8. Для перехода в динамике z_2 необходима перебрать число UK , количество вершин, которые мы возьмем в множество из поддерева вершины U , от 1 до K и прибавить к ответу $z[U][UK] + z_2[V][Y][K - UK]$, где Y — это следующий за U сын вершины V .

День седьмой (22.02.2013 г.) Контеcт Куликова Егора Юрьевича

Об авторе...

Куликов Егор Юрьевич, родился 28 июня 1985 года в городе Ярославле.

Учился в школе №11 города Люберцы с 1992 - 1993г., в школе №. 84 города Ярославль с 1993 - 1998 г., и в школе №33 им. Карла Маркса города Ярославль с 1998 - 2002 г.

В 2007 году окончил Механико - математический факультет, кафедра Дискретной математики Московского государственного университета 2005 - 2007 г. работал в Лаборатории вычислительных методов МГУ им. Ломоносова.

С 2007-2009 г. работал в компании Google в Санкт-Петербурге.

С 2012 года работает в компании Яндекс.



Основные достижения:

- Победитель Google CodeJam 2010;
В составе команды МГУ x13 победитель Чемпионата России 2006;
- На январь 2012 года — 8е место в рейтинге TopCoder и 3е в рейтинге Codeforces.

Теоретический материал. Суффиксный автомат

Определение

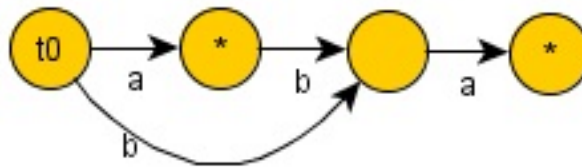
Суффиксный автомат для строки s — это такой ориентированный ациклический граф, что:

1. Каждое ребро помечено буквой строки s . Не из какой вершины не выходят ребра с одинаковыми метками.
2. Он имеет исток t_0 .

3. Любому пути из стока можно поставить соответствие строку из меток на ребрах. Тогда каждая подстрока s встретится ровно один раз, а каждый такой путь будет соответствовать какой-либо подстроке s .

Вершина автомата называется **терминальной** если любой путь из t_0 , заканчивающийся в этой вершине, является суффиксом s .

Например, суффиксный автомат для строки **aba** выглядит так (* помечены терминальные вершины):



Построение

Построим автомат и докажем, что у него будет линейное от размера строки число вершин и ребер.

Рассмотрим любую непустую подстроку t строки s . Рассмотрим все позиции в строке s , в которых заканчивается вхождение строки t . Назовем множество подобных событий $endPos(t)$. Тогда все подстроки s распадутся на классы эквивалентности по равенству множеств $endPos$ (при этом вершина t_0 будет соответствовать пустой подстроке с множеством $endPos$ равным $[-1 \dots length(s) - 1]$). Это и будут вершины нашего автомата.

Рассмотрим две подстроки u и w строки s (длина u не более длины w). Тогда можно заметить, что если u не является суффиксом w , то $endPos(u)$ и $endPos(w)$ не пересекаются, в противном случае $endPos(w) \subseteq endPos(u)$. Таким образом один класс эквивалентности будет состоять из какой-то строки w и всех ее суффиксов больше какого-то размера. Тогда назовем длину этой строки длиной соответствующей вершины автомата, саму строку — строкой данной вершины, а вершину, которой соответствует самый большой суффикс w , не входящий в тот же класс эквивалентности — суффиксной ссылкой данной вершины. Тогда заметим, что суффиксные ссылки образуют дерево с корнем в t_0 , при этом если объединить все классы эквивалентности, соответствующие вершинам на пути от данной вершины v к корню t_0 , то они будут содержать все суффиксы строки, соответствующей v и только их.

Перейдем непосредственно к алгоритму. Алгоритм будет онлайн-овым, то есть за один шаг мы из автомата для строки $s[0 \dots i]$ будем получать автомат для строки $s[0 \dots i + 1]$. В каждой вершине мы будем хранить ее длину, суффиксную ссылку и исходящие ребра. Изначально автомат

состоит из одной вершины t_0 с длиной 0, суффиксной ссылкой *null* и без исходящих ребер. Так же мы будем хранить последнюю вершину *last*, которое соответствует всей уже построенной строке. Шаг алгоритма по добавлению буквы c будет такой:

- Создаем новую вершину *cur* с длиной, равной длине *last* + 1.
- Начиная с *last* идем по суффиксным ссылкам, пока не дойдем до исходящего ребра с меткой c . Создаем в каждой из них исходящее ребро с меткой c в вершину *cur*.
- Если мы в какой-то момент дошли до вершины p , из которой есть ребро с меткой c в вершину q , то возможны 2 варианта:
 - Если длина вершины q — это длина вершины $p + 1$, тогда q — это суффиксная ссылка вершины *cur* и данный шаг алгоритма окончен.
 - Иначе построим вершину *clone* с длиной, равной длине $p + 1$, с исходящими ребрами в те же вершины и с теми же метками, что у q , суффиксной ссылкой в ту же вершину, что и q . Теперь установим суффиксные ссылки из q и *cur* в *clone*, а так же пойдём из p по суффиксным ссылкам, и пока будет существовать ребро с меткой c и значением q по этой метке, перенаправим это ребро в *clone*.
- Если же мы так и не нашли ребра с меткой c , то установим суффиксную ссылку *cur* в вершину t_0 .
- В любом случае *last* теперь будет соответствовать *cur*.

Докажем корректность алгоритма. Обозначим за s строку до добавления символа c . Назовем ребро (p, q) сплошным, если длина вершины p на один меньше длины вершины q . В противном случае ребро будет не сплошным.

Состояние *cur* должно создаваться, так как ему соответствует строка $s + c$ ($endPos(s + c) = length(s)$, подобного множества, очевидно, ранее не было). В эту вершину мы проведем ребра из всех тех суффиксов w строки s , для которых строки $w + c$ не встречалось в качестве подстроки в s . Пусть теперь w — это такой максимальный суффикс s для которого $w + c$ уже встречалось в качестве подстроки в s . Если переход из вершины, соответствующей w по c сплошной (то есть $w + c$ — строка, соответствующая вершине, куда идет этот переход), то в множество *endPos* для $w + c$ и всех его суффиксов войдет $length(s)$. В противном случае произойдет расщепление — $endPos(w + c)$ изменится, в то время, как $endPos(q)$ —

нет (где q — строка, соответствующая вершине, в которую идет переход из w по c). Этому соответствует шаг алгоритма с созданием вершины *clone*.

Докажем, что в алгоритме у нас линейное число операций. Очевидно, что у нас линейное число вершин (так как на каждом шагу мы добавляем не более $2x$). Докажем, что число ребер так же линейно. Заметим, что сплошные ребра у нас образуют остовное дерево, а значит их меньше, чем вершин. Любому же не сплошному ребру соответствует суффикс — а именно если мы рассмотрим для несплошного ребра (p, q) самый длинный путь из t_0 до p и самый длинный путь из q в терминал, то все переходы в нем будут по сплошным ребрам, а в сумме получившийся пусть из t_0 в терминал будет соответствовать какому-то суффиксу. Так как у строки не более длины различных суффиксов, то и несплошных ребер линейное число. На самом деле легко видеть, что количество вершин не более $2n - 1$, а ребер — не более $3n - 3$, где n — длина s .

Теперь осталось доказать, что мы линейное число раз перенаправим ребро (так как все остальные части алгоритма либо создают вершину, либо ребро). Для этого заметим, что будет монотонно увеличиваться позиция последнего вхождения суффиксной ссылки от суффиксной ссылки от $last(q)$ на каждом подобном шагу алгоритма. То есть всего это произойдет не более $length(s)$ раз.

Реализация

```

1 public class SuffixAutomaton {
2     public final int[] length;
3     public final int[] link;
4     public final int[] first;
5     public final int[] next;
6     public final int[] to;
7     public final int[] label;
8     public int size;
9     public int last;
10
11     public SuffixAutomaton(CharSequence s) {
12         int count = s.size();
13         length = new int[2 * count];
14         link = new int[2 * count];
15         first = new int[2 * count];
16         next = new int[4 * count];
17         label = new int[4 * count];
18         to = new int[4 * count];
19         Arrays.fill(first, -1);
20         link[0] = -1;
21         size = 1;
22         int edgeSize = 0;
23         last = 0;
24         for (int i = 0; i < s.length(); i++) {
25             int c = s.charAt(i);

```

```

26     int current = size++;
27     length[current] = length[last] + 1;
28     for (int previous = last; ; previous = link[previous]) {
29         if (previous == -1) {
30             link[current] = 0;
31             break;
32         }
33         int index = findEdge(previous, c);
34         if (index != -1) {
35             int curLink = to[index];
36             if (length[previous] + 1 == length[curLink])
37                 link[current] = curLink;
38             else {
39                 int clone = size++;
40                 length[clone] = length[previous] + 1;
41                 link[clone] = link[curLink];
42                 int linkEdge = first[curLink];
43                 while (linkEdge != -1) {
44                     next[edgeSize] = first[clone];
45                     first[clone] = edgeSize;
46                     label[edgeSize] = label[linkEdge];
47                     to[edgeSize++] = to[linkEdge];
48                     linkEdge = next[linkEdge];
49                 }
50                 for (; previous != -1; previous = link[previous]) {
51                     int edge = findEdge(previous, c);
52                     if (edge == -1 || to[edge] != curLink)
53                         break;
54                     to[edge] = clone;
55                 }
56                 link[current] = link[curLink] = clone;
57             }
58             break;
59         }
60         next[edgeSize] = first[previous];
61         first[previous] = edgeSize;
62         label[edgeSize] = c;
63         to[edgeSize++] = current;
64     }
65     last = current;
66 }
67 }
68
69 public int findEdge(int vertex, int label) {
70     int edge = first[vertex];
71     while (edge != -1) {
72         if (this.label[edge] == label)
73             return edge;
74         edge = next[edge];
75     }
76     return -1;
77 }
78 }

```

Пример использования

Пусть нам надо посчитать число различных подстрок строки s . Заметим, что в суффиксном автомате по строке s — это просто количество различных путей. Количество путей в ациклическом графе можно найти динамически — это $1 +$ сумма значений динамики для всех вершин, в которые из данной идет ребро.

Связь с суффиксным деревом

Докажем, что если мы возьмем строку t , являющуюся развернутой строкой s , построим по ней суффиксный автомат, а затем развернем в нем суффиксные ссылки, то мы построим суффиксное дерево строки s . И правда, суффиксная ссылка ведет в такой наибольший суффикс, который является отдельным состоянием. А значит в развернутой строке он ведет в такой наибольший префикс, а это и есть вершина, из которой должно быть ребро в данную.

Давайте так же заметим, что если мы развернем сплошные ребра в автомате, то мы получим суффиксные ссылки. Таким образом, мы по суффиксному автомату по строке t можем построить суффиксное дерево для строки s . Для того, чтобы совершить обратное преобразование (нам нужно получить дополнительно несплошные ребра) мы введем дополнительно определение расширяющей ссылки — это ребро, обратное суффиксной ссылке. Тогда для каждой вершины суффиксного дерева, начиная с листьев, надо рассмотреть ее детей и скопировать расширяющую ссылку из ребенка (если в данной вершине по данному символу ее еще нет) — эти дополнительные ребра и будут несплошными ребрами в суффиксном автомате.

Задачи и разборы

Задача A. Chords

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 s
Ограничение по памяти:	256 Mb

Consider a circle. There are $2N$ different points marked on it and numbered with integer numbers in range from 1 to N so that each number from the interval has two points corresponding to it.

Points with the same numbers are connected by segments. Thus we've got N chords. Furthermore, the chords are numbered as well: the chord number

“ i ” connects the two different points with number “ i ”. Apparently some of the chords may intersect. Now it would be nice to find out for each single chord the number of chords it intersects.

Формат входного файла

In the first line of the input file there is a single integer number N ($1 \leq N \leq 10^5$). In the next line there are $2N$ integers in range from 1 to N — the numbers assigned to the points in the order of traversal. Each number is written exactly twice. All the numbers in the line are separated with spaces.

Формат выходного файла

The output file is supposed to contain exactly N lines: on the i -th line write the number of chords that i -th chord intersects.

Пример

stdin	stdout
5	2
1 2 3 1 4 2 5 5 3 4	3
	3
	2
	0

Разбор задачи A. Chords

Перенумеруем хорды так, чтобы левые концы хорд шли слева направо. Теперь пойдем по концам хорд слева направо, записывая левые концы в упорядоченное множество, а правые — удаляя. После удаления мы можем узнать количество хорд, для которых левый конец лежит между концами текущей хорды, а правый — нет. Это хорды с номерами большими, чем у текущей хорды, лежащими в нашем множестве. Осуществив аналогичный процесс справа налево, мы также получим количество хорд, у которых между концами данной лежит правый конец. Просуммировав эти два значения получим ответ

Задача B. Cyclic suffixes

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 s
 Ограничение по памяти: 256 Mb

Consider a string $S = s_1s_2s_3 \dots s_{n-1}s_n$ over an alphabet Σ . A *cyclic*

expansion of order m over a string S is the string $s_1s_2s_3 \dots s_{n-1}s_ns_1s_2 \dots$ of m characters; that is, we concatenate instances of S until the resulting string has sufficient length, and then take the first m characters as the result.

Let *cyclic string* \tilde{S} be the infinite cyclic expansion over a string S .

Take a look at the suffixes of a cyclic string \tilde{S} . Obviously, there are no more than $|S|$ different ones among them: the $(n+1)$ -th one is equal to the first (the cyclic string itself), the $(n+2)$ -th is equal to the second, and so on. However, there may be even less different suffixes. For example, if $S = \mathbf{abab}$, the first suffixes of \tilde{S} are:

$$\begin{aligned}\tilde{S}_1 &= \mathbf{ababababab} \dots \\ \tilde{S}_2 &= \mathbf{bababababa} \dots \\ \tilde{S}_3 &= \mathbf{ababababab} \dots \\ \tilde{S}_4 &= \mathbf{bababababa} \dots\end{aligned}$$

etc. Here, only two different suffixes can be found, while $|S| = 4$.

Let us order the first $|S|$ suffixes of \tilde{S} lexicographically. If two suffixes are equal, the one with the lower index comes first. We are now interested in the following problem: what is the position of our original string \tilde{S} after sorting?

The following example shows the ordering for $S = \mathbf{cabcab}$.

$$\begin{aligned}(1) \quad \tilde{S}_2 &= \mathbf{abcabcbca} \dots \\ (2) \quad \tilde{S}_5 &= \mathbf{abcabcbca} \dots \\ (3) \quad \tilde{S}_3 &= \mathbf{bcabcbcab} \dots \\ (4) \quad \tilde{S}_6 &= \mathbf{bcabcbcab} \dots \\ (5) \quad \tilde{S}_1 &= \mathbf{cabcbcabcb} \dots \\ (6) \quad \tilde{S}_4 &= \mathbf{cabcbcabcb} \dots\end{aligned}$$

Here, the position of $\tilde{S} = \tilde{S}_1$ is 5.

Given a string S of length n , find the number of \tilde{S} in the specified ordering.

Формат входного файла

On the first line of the input file, a string S ($1 \leq |S| \leq 1\,000\,000$) is given. The input consists of lowercase Latin letters only.

Формат выходного файла

Output the only number on a line by itself — the number of \tilde{S} in the specified ordering of its first $|S|$ suffixes.

Примеры

stdin	stdout
abracadabra	3
cabcab	5

Задача C. A Coloring Game

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 2 s
 Ограничение по памяти: 256 Mb

Two players play a graph coloring game. They make moves in turn, first player moves first. Initially they take some undirected graph. At each move, a player can color an uncolored vertex with either white or black color (each player can use any color, possibly different at different turns). It's not allowed to color two adjacent vertices with the same color. A player that can't move loses.

After playing this game for some time, they decided to study it. For a start, they've decided to study very simple kind of graph — a chain. A chain consists of N vertices, v_1, v_2, \dots, v_N , and $N - 1$ edges, connecting v_1 with v_2 , v_2 with v_3 , \dots , v_{N-1} with v_N .

Given a position in this game, and assuming both players play optimally, who will win?

Формат входного файла

The first line of the input file contains the integer N , $1 \leq N \leq 100\,000$.

The second line of the input file describes the current position. It contains N digits without spaces. i^{th} digit describes the color of vertex v_i : 0 — uncolored, 1 — black, 2 — white. No two vertices of the same color are adjacent.

Формат выходного файла

On the only line of the output file, print “FIRST” (without quotes) if the player moving first in that position wins the game, and “SECOND” (without quotes) otherwise.

Пример

stdin	stdout
5 00100	SECOND
4 1020	FIRST

Разбор задачи C. A Coloring Game

Посчитаем числа Гранди для позиций вида $x0 \dots 0x$ и $x0 \dots 0y$ (где x, y и 0 попарно не равны). Докажем, что число Гранди первой позиции — 1, а второй — 0 по индукции по числу нулей. База (0 нулей для второго случая и один ноль для первого) очевидна. В переходе из первой позиции мы можем получить $0 \text{ xor } 0$ или $1 \text{ xor } 1$, то есть число Гранди равно 1. Аналогично для второго случая мы можем получить $0 \text{ xor } 1$ или $1 \text{ xor } 0$.

Теперь осталось разобрать отрезки нулей, один из концов которых — не 0. Если такой конец ровно один, то так же по индукции легко показать, что число Гранди будет равно числу нулей, а если их два — то из предыдущего утверждения следует, что число Гранди равно числу нулей по модулю 2.

Задача D. Hippopotamus

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 s
 Ограничение по памяти: 256 Мб

You think that your roof looks unpretty. So you opt for a new one, consisting of n consecutive long narrow boards. You have two types of boards: wooden ones and iron ones, giving you an amazing total of 2^n possible roofs.

But the safety should not be left aside. Having considered the weight and the cruising speed of a falling hippopotamus, you decide to have at least k iron boards among every m consecutive boards.

How many possibilities do you have?

Формат входного файла

The input file contains three integers, n , m and k , separated by spaces and/or line breaks. $1 \leq n \leq 60$, $1 \leq m \leq 15$, $0 \leq k \leq m \leq n$.

Формат выходного файла

Output the number of possibilities.

Пример

stdin	stdout
10 2 1	144
5 5 2	26
3 2 2	1

Разбор задачи D. Hippopotamus

Давайте решим задачу динамикой со следующим состоянием — (кол-во уже обработанных планок, битовая маска последних $m - 1$ планок). При этом бит включен, если соответствующая планка железная. Начинать мы можем с количества планок равного нулю, при этом в состоянии 0 все биты включены и ответ будет 1, а во всех остальных с 0 в первом параметре — 0.

Задача E. False RSA

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 s
 Ограничение по памяти: 256 Mb

Roman, Serge and Andrew has decided to upgrade the famous RSA encryption algorithm. They think that the limitation that modulo n used in RSA must be a product of two distinct primes is redundant. Instead they plan to use n which is the product of k -th powers of two distinct primes: $n = p^k q^k$.

However, Nick pointed out that besides all other mathematical problems, the scheme may be easier to crack. That is — it is difficult to factorize n which is a product of two distinct primes, because it has exactly one non-trivial factorization. On the other hand, in case $n = p^k q^k$ there can be more different non-trivial factorizations. For example, $100 = 2^2 5^2$ has eight non-trivial factorizations: $100 = 2 \cdot 50$, $100 = 2 \cdot 2 \cdot 25$, $100 = 2 \cdot 2 \cdot 5 \cdot 5$, $100 = 2 \cdot 5 \cdot 10$, $100 = 4 \cdot 25$, $100 = 4 \cdot 5 \cdot 5$, $100 = 5 \cdot 20$ and $100 = 10 \cdot 10$.

Now Roman, Serge and Andrew wonder — given $n = p^k q^k$, how many different non-trivial factorizations of n are there?

Формат входного файла

The input file contains one integer number n ($6 \leq n \leq 10^{18}$, it is guaranteed that $n = p^k q^k$ for different primes p and q and integer $k > 0$).

Формат выходного файла

Output one integer number — the number of different non-trivial factorizations of n .

Пример

stdin	stdout
6	1
100	8

Разбор задачи E. False RSA

Заметим, что ответ зависит только от значения k . Найдем сначала k .

Для этого заметим, что если $p < q$, то $p^k < \sqrt{(n)}$, то есть не более 10^9 . Таким образом если $k > 1$, то $p < \sqrt{(10^9)}$, и мы можем просто перебрать p . Если мы не нашли делителя, то $k = 1$.

Заметим, что $k \leq \log_6 10^{18} < 24$. Будем считать динамику, где состоянием будет (a = текущая степень p , b = текущая степень q , c = степень p в последнем делителе, d = степень q в последнем делителе), а значением — количество разложений числа $p^a q^b$, где все делители имеют либо p в степени меньшей, чем c , либо p в степени c , а q — в степени не более, чем d . Реализация этой динамики остается читателю в качестве упражнения.

Задача F. Perspective

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 s
Ограничение по памяти:	256 Mb

Breaking news! A Russian billionaire has bought a yet undisclosed NBA team. He's planning to invest huge effort and money into making that team the best. And in fact he's been very specific about the expected result: the first place.

Being his advisor, you need to determine whether it's possible for your team to finish first in its division or not.

More formally, the NBA regular season is organized as follows: all teams play some games, in each game one team wins and one team loses. Teams are grouped into divisions, some games are between the teams in the same division, and some are between the teams in different divisions.

Given the current score and the total number of remaining games for each team of your division, and the number of remaining games between each pair of teams in your division, determine if it's possible for your team to score at least as much wins as any other team in your division.

Формат входного файла

The first line of the input file contains N ($2 \leq N \leq 20$) — the number of teams in your division. They are numbered from 1 to N , your team has number 1.

The second line of the input file contains N integers w_1, w_2, \dots, w_N , where w_i is the total number of games that i^{th} team has won to the moment.

The third line of the input file contains N integers r_1, r_2, \dots, r_N , where r_i is the total number of remaining games for the i^{th} team (including the games inside the division).

The next N lines contain N integers each. The j^{th} integer in the i^{th} line of those contains a_{ij} — the number of games remaining between teams i and j . It is always true that $a_{ij} = a_{ji}$ and $a_{ii} = 0$, for all i $\sum_j a_{ij} \leq r_i$.

All the numbers in the input file are non-negative and don't exceed 10 000.

Формат выходного файла

On the only line of output, print “YES” (without quotes) if it's possible for the team 1 to score at least as much wins as any other team of its division, and “NO” (without quotes) otherwise.

Пример

stdin	stdout
3 1 2 2 1 1 1 0 0 0 0 0 0 0 0 0	YES
3 1 2 2 1 1 1 0 0 0 0 0 1 0 1 0	NO

Разбор задачи F. Perspective

Считаем, что наша команда выиграет все матчи. Тогда для каждой из остальных команд мы знаем, сколько максимум матчей она может выиграть. Построим граф, вершинами в котором будут исток, сток, возможные пары команд, кроме нашей и вершины — одиночные команды. Тогда из истока проведем в каждую возможную пару ребро пропускной способности, равной числу игр между командами в этой паре, из каждой возможной пары команд — в команды этой пары с бесконечной пропускной способностью, а из каждой команды — в исток с пропускной способностью, равной максимальному числу побед, которое эта команда может одержать, не обогнав при этом нашу команду. Тогда наша команда может выиграть, если в этом графе существует поток, равный числу оставшихся матчей без участия нашей команды.

Задача G. Circular Railway

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 s
Ограничение по памяти:	256 Mb

There are L stations along a circular railway, numbered 1 through L . Trains travel in both directions, and take 1 minute to get from a station to the neighbouring one (i.e., between 1st and 2nd, between 2nd and 3rd, ..., between $(L - 1)$ -th and L -th and between L -th and 1-st).

There are n employee's houses along the railway, and n offices, each house or office located near a railway station. You are to establish a one-to-one correspondence between houses and offices in such a way that total travel time (sum of travel times of each employee) is minimized.

Формат входного файла

The first line of the input file contains two integer numbers, n and L ($1 \leq n \leq 50000$, $2 \leq L \leq 10^9$). The second line contains n locations of the employee's houses, and the third line contains n locations of the offices. Each location is an integer number between 1 and L . Some houses or offices or both can be located at the same railway station.

Формат выходного файла

Output the minimal total travel time followed by the description of the one-to-one correspondence. The description should be represented by n numbers

(one for each employee, ordered as in the input), denoting the 1-based index of the office assigned to the corresponding employee.

Пример

stdin	stdout
3 15 1 2 10 11 12 13	9 2 3 1
4 12 2 5 8 11 6 9 12 3	4 4 1 2 3

Разбор задачи G. Circular Railway

Оптимальное решение: по каждому перегону люди будут двигаться только в одну сторону. Иначе, если есть два человека, которые двигаются по данному перегону в разные стороны, мы можем поменять местами их работы, после чего ответ уменьшится на 2. Кроме того, если мы обозначим за a_i количество людей,двигающихся по перегону между станциями i и $i+1$ слева направо (отрицательное число, если движение идет справа налево, a_n — количество людей,двигающихся между станциями n и 1), то все числа определяются значением a_1 (а именно — $a_{i+1} - a_i =$ количеству людей, живущих на $i+1$ станции минус количество людей там работающих). Таким образом, нам надо подобрать такое a_1 , при котором сумма модулей a_i минимальна. Очевидно это достигается, когда число отрицательных и положительных a_i различается не более, чем на количество a_i равных нулю. Посчитаем значения a_i для a_1 равного нулю, отсортируем и вычтем из всех медиану.

Задача H. SETI

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 seconds
 Ограничение по памяти: 256 s

Amateur astronomers Tom and Bob try to find radio broadcasts of extraterrestrial civilizations in the air. Recently they received some strange signal and represented it as a word consisting of small letters of the English alphabet. Now they wish to decode the signal. But they do not know what to start with.

They think that the extraterrestrial message consists of words, but they cannot identify them. Tom and Bob call a subword of the message a *potential word* if it has at least two non-overlapping occurrences in the message.

For example, if the message is “**abacabacaba**”, “**abac**” is a potential word, but “**acaba**” is not because two of its occurrences overlap.

Given a message m help Tom and Bob to find the number of potential words in it.

Формат входного файла

Input file contains one string that consists of small letters of the English alphabet. The length of the message doesn't exceed 10 000.

Формат выходного файла

Output one integer number — the number of potential words in a message.

Пример

stdin	stdout
abacabacaba	15

Разбор задачи Н. SETI

Построим суффиксный автомат для строки сообщения. Кроме того, для каждой вершины так же найдем первую и последнюю позицию из ее множества $endPos$. Тогда потенциальное слово соответствует этой вершине тогда и только тогда, когда его длина не превышает разницу между этими значениями. Но данной вершине соответствуют слова длины не более ее длины и более длины ее суффиксной ссылки. Пройдемся по всем вершинам автомата и посчитаем их количество.

Задача I. 2-3 Trees

Вход: stdin
 Выход: stdout
 Ограничение по времени: 2 s
 Ограничение по памяти: 256 Mb

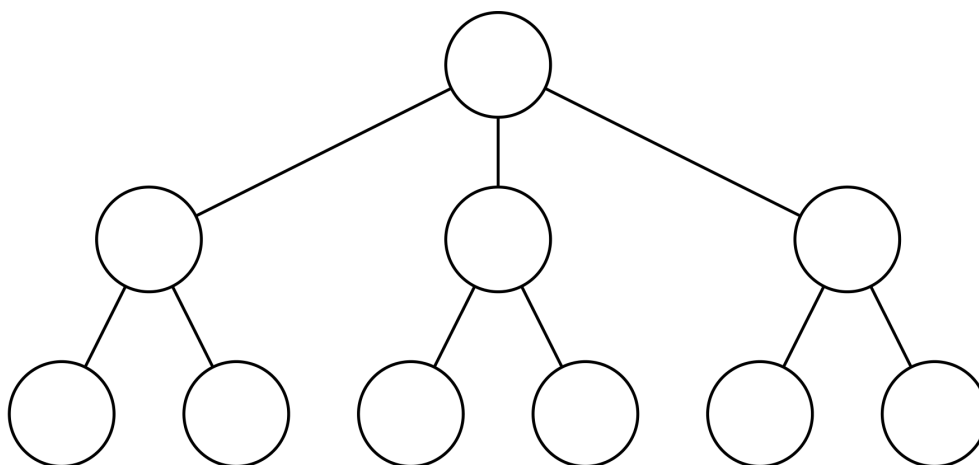
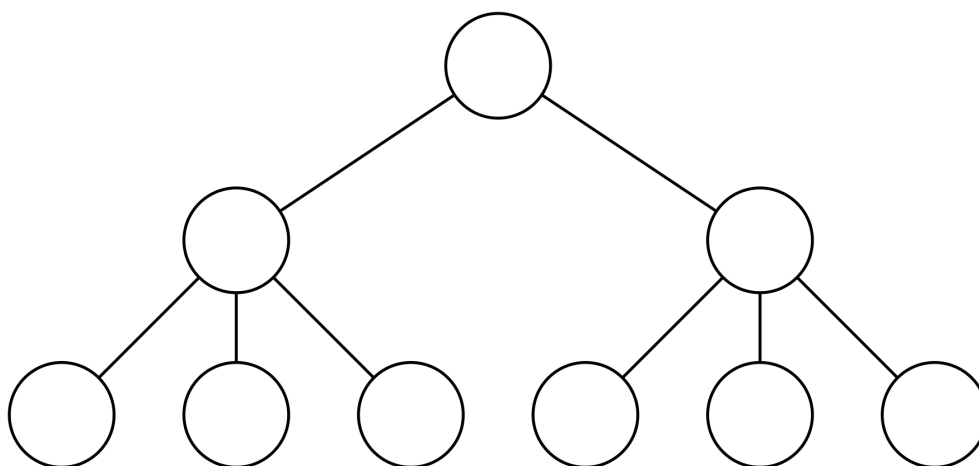
2-3 tree is an elegant data structure invented by John Hopcroft. It is designed to implement the same functionality as the binary search tree. 2-3 tree is an ordered rooted tree with the following properties:

- the root and each internal vertex have either 2 or 3 children;

- the distance from the root to any leaf of the tree is the same.

The only exception is the tree that contains exactly one vertex — in this case the root of the tree is the only vertex, and it is simultaneously a leaf, i.e. has no children. The main idea of the described properties is that the tree with l leaves has the height $O(\log l)$.

Given the number of leaves l there can be several valid 2-3 trees that have l leaves. For example, the picture below shows the two possible 2-3 trees with exactly 6 leaves.



Given l find the number of different 2-3 trees that have l leaves. Since this number can be quite large, output it modulo r .

Формат входного файла

Input file contains two integer numbers: l and r ($1 \leq l \leq 5\,000$, $1 \leq r \leq 10^9$).

Формат выходного файла

Output one number — the number of different 2-3 trees with exactly l leaves modulo r .

Пример

stdin	stdout
6 1000000000	2
7 1000000000	3

Разбор задачи I. 2-3 Trees

Посчитаем динамикой количество способов разбить $k \leq l$ вершин на p частей, в каждой из которых 2 или 3 последовательных вершины. Теперь посчитаем динамикой число 2-3 деревьев с $v \leq l$ листьями — для всех p просуммируем количество способов разбить v вершин на p частей умноженное на число 2-3 деревьев с p листьями.

День восьмой (23.02.2013 г.)

Контеcт Гольдштейна Виталия Борисовича

Об авторе...

Гольдштейн Виталий Борисович, ассистент и аспирант Московского Физико-Технического Института. Член научного комитета всероссийской олимпиады школьников по информатике и сборов по подготовке к международной олимпиаде школьников. Стажер компании Google (зима-весна 2008). Завуч Летней Компьютерной Школы (август, 2009) и Летней Компьютерной Школы (зима, 2009-2010). Член жюри Московского и Саратовского четвертьфинала ACM ICPC, член жюри полуфинала NEERC ACM ICPC. Работал в компании Яндекс. Лектор курса “Алгоритмы и структуры данных” базовой кафедры Яндекса “Анализ данных” на факультете Инноваций и высоких технологий в МФТИ. Член тренерской группы проекта “Яндекс-тренировки” в Москве.



С января работает в компании Google в Москве.

Основные достижения:

- серебряная медаль чемпионата мира по программированию среди студентов (ACM ICPC Tokyo 2007);
- участник финалов TopCoder Open, TopCoder Collegiate Open, Global Google Code Jam, Google Code Jam Europe;
- тренер призеров (4-е место, золотая медаль) чемпионата мира по спортивному программированию ACM-ICPC World Finals 2009, Швеция, Стокгольм.

Теоретический материал. Кратчайшие пути в графах

Алгоритм Дейкстры

Алгоритм Дейкстры поддерживает три множества вершин:

- белые – расстояние до них неизвестно;
- черные – кратчайшее расстояние до них найдено абсолютно точно;
- серые – до них известно расстояние, использующее в качестве промежуточных только черные.

Основной теоремой, доказывающей алгоритм, утверждается, что серая вершина с минимальным расстоянием может быть помещена во множество черных, то есть кратчайшее расстояние до нее найдено абсолютно верно. Однако для доказательства этого факта используется три утверждения (одно явно, а два неявно):

- вес пути не уменьшается при добавление ребер (можно сказать, что нет ребер отрицательного веса);
- выгодно продолжать кратчайший путь. То есть не имеет смысла идти сначала по более длинному пути, чтобы потом дополнить его и суммарный вес стал меньше;
- все кратчайшие пути одинаково полезны, то есть нет смысла предпочитать один кратчайший путь другому.

Второй и третий факты очевидны в случае, когда вес пути это сумма весов ребер. Однако бывают и более сложные задачи, использующие такие весовые функции как:

- вес максимального ребра;
- среднее арифметическое ребер;
- сумма двух максимальных ребер.

Задачи использующие первый вариант весовых функций возможно решить с помощью алгоритма Дейкстры, а второе и третье уже нет. Формально, если обозначить вес пути за $P(W)$, то для применимости алгоритма Дейкстры должны выполняться следующие два свойства:

$$P(A) \leq P(A + e)$$

$$P(A) \leq P(B) \Rightarrow P(A + e) \leq P(B + e)$$

A и B тут пути, а e – ребро. Для среднего арифметического не выполнен уже первый пункт, а для суммы двух максимальных ребер только второй.

Классические примеры задач, которые можно решить с помощью алгоритма Дейкстры – это кратчайший путь с учетом пробок и кратчайший маршрут с учетом расписания общественного транспорта.

Решим задачу нахождения кратчайшего пути между двумя заданными вершинами s и t , где весом пути является сумма двух максимальных ребер. Как уже было сказано, это невозможно решить с помощью алгоритма

Дейкстры в явном виде. Однако, мы можем запустить алгоритм Дейкстры с двух концов: из начала пути и из конца пути по транспонированному графу, с весовой функцией — максимальное ребро пути. Тогда мы посчитаем $d[v]$ — кратчайшее расстояние из вершины s до v и $e[v]$ — кратчайшее расстояние от v до t . С таким предподсчетом мы сможем решить задачу нахождения кратчайшего пути в графе из вершины s в вершину t с весовой функцией — сумма двух максимальных ребер, где максимальным ребром пути является ребро из u в v . Если вес этого ребра w больше либо равен $\max(d[u], e[v])$, то такой путь существует и его вес равен $w + \max(d[u], e[v])$. В противном случае такого пути не существует. Ответом на задачу будет минимум по всем существующим путям.

Алгоритм Форда-Беллмана

Данный алгоритм является несложным динамическим программированием. $D[k][v]$ — кратчайшее расстояние из стартовой вершины s до вершины v за ровно k ребер. Для пересчета массива $D[k + 1]$ необходимо знание только предыдущего массива $D[k]$. Если в графе отсутствуют циклы отрицательного веса, то кратчайший путь состоит не более, чем из $N - 1$ ребра, где N равно количеству вершин. Для оптимизации объема памяти можно хранить только две последние строки, однако эта оптимизация сделает невозможным восстановление пути.

Аналогично можно вычислить $E[k][v]$ — кратчайшее расстояние из стартовой вершины s до вершины v не более, чем за k ребер. Классическим алгоритмом Форда-Беллмана является расчет данных значений в одном массиве. То есть просто $N - 1$ итерация обновления одного и того же массива $W[v]$. При таком расчете нельзя точно сказать какое именно значение хранится в массиве $W[v]$ после k итераций. Можно только сказать, что $W[v] \leq D[k][v]$. Это не позволяет контролировать количество ребер в пути и не позволяет решать некоторые задачи.

Если в алгоритме Форда-Беллмана на дополнительной итерации номер N произойдет хотя бы одно изменение массива W , то это признак, что в графе есть цикл отрицательного веса. Цикл отрицательного веса можно выделить двигаясь по “дереву” кратчайших путей из вершины, в которой произошло изменение. Нет гарантии, что эта вершина входит в цикл, однако путь до нее состоит более, чем из $N - 1$ ребра, а значит содержит цикл.

Задачи и разборы

Задача А. Красная Шапочка

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

В непроходимом лесу имеется N полянок и M тропинок между ними. Каждая тропинка соединяет две различные полянки. Две полянки могут быть соединены несколькими тропинками.

На двух разных полянках живут Красная Шапочка и ее бабушка. Домик Красной Шапочки находится на полянке с номером 1, а домик бабушки — на полянке с номером N . Красная Шапочка хорошо ориентируется в лесу и знает, какое минимальное время ей потребуется для прохождения каждой тропинки. Когда Красная Шапочка идет по лесу, она переходит с тропинки на тропинку только на полянках. На каждой полянке есть укрытие, в котором Красная Шапочка может спрятаться на некоторое время.

В этом же лесу живет Волк. Время, за которое Волк пробегает какую-либо тропинку, может отличаться от времени, за которое по ней проходит Красная Шапочка. Кроме того, если Волк пробегает по одной и той же тропинке несколько раз, то каждый раз он может тратить на это разное время.

С края полянки, где живет Красная Шапочка, Волк увидел, что она собирается нести пирожки бабушке и побежал по тропинкам привычного ему пути от дома Красной Шапочки к дому бабушки. Волк начинает бежать от домика Красной Шапочки в тот момент, когда она решила выйти из дома, его путь заканчивается как только он окажется на полянке с домиком бабушки. Ни на одной полянке Волк не задерживается.

Чтобы застать бабушку в целости и сохранности, Красной Шапочке необходимо обогнать Волка. При этом ей нельзя оказаться с Волком на одной тропинке, даже если Волк уже покидает ее, а она только появляется на ней, или наоборот. Чтобы избежать встречи с Волком на полянке, Красная Шапочка использует имеющееся там укрытие. Красной Шапочке нельзя появляться на полянке одновременно с Волком или покидать укрытие на полянке в тот момент, когда на ней появляется Волк. При необходимости Красная Шапочка может идти по тропинке дольше минимально возможного времени, а также выйти из дома позже, чем она исходно решила.

Необходимо написать программу, которая поможет Красной Шапочке добраться к бабушке раньше Волка, если известна последовательность тропинок, по которым побежал Волк.

Формат входного файла

Первая строка входного файла содержит числа N , M и K ($2 \leq N \leq 2000$, $1 \leq M \leq 100\,000$, $1 \leq K \leq 100\,000$). Следующие M строк содержат по три числа: B_i , E_i — номера полянок, которые соединяет i -я тропинка, и T_i — минимальное время, за которое Красная Шапочка может по ней пройти ($1 \leq T_i \leq 10\,000$).

В следующих K строках находится последовательное описание пути Волка, по два числа в строке: P_i — номер тропинки, по которой он побежит, и V_i — время, которое он на это затратит ($1 \leq V_i \leq 10\,000$). Путь волка всегда начинается на полянке 1 и заканчивается на полянке N .

Все числа во входном файле целые и в пределах одной строки разделены пробелами.

Формат выходного файла

В том случае, если Красная Шапочка не может добраться до домика бабушки быстрее Волка, выходной файл должен содержать слово “NO”.

Если Красная Шапочка сможет добраться до домика бабушки быстрее волка, в первой строке выходного файла должно быть слово “YES”. Во второй строке в этом случае должно содержаться число тропинок в пути Красной Шапочки. В третью строку следует вывести номера тропинок в том порядке, в котором Красная Шапочка должна по ним пройти. Числа должны быть разделены пробелами.

Информацию о времени прохождения по тропинкам и остановках на полянках в выходной файл выводить не нужно.

Пример

stdin	stdout
4 4 5 1 3 6 1 2 2 2 3 2 3 4 1 2 1 2 2 2 1 3 4 4 1	YES 2 1 4
4 3 4 1 2 2 2 3 1 2 4 3 1 2 2 1 2 2 3 5	NO

Разбор задачи А. Красная Шапочка

Эта задача является одним из примеров графа с динамическими ребрами. В данном случае некоторые ребра исчезают из графа (когда по ним проходит волк). Первым этапом решения задачи является определение про каждое ребро, интервалы времени, когда по нему нельзя ходить красной шапочки. Если волк входит на ребро во время L , а выходит из ребра во время R , то красной шапочке нельзя начинать движение по этому ребру в моменты времени из отрезка $[L - T, R]$, где T равно времени прохождения данного ребра Красной Шапочкой. После этого задача решается алгоритмом Дейкстры, в котором вес ребра определяется динамически. Если Красная Шапочка хочет воспользоваться ребром, которое сейчас недоступно, то его вес нужно увеличить на время до его появления (по аналогии с путешествием на общественном транспорте, когда к длине ребра добавляется время ожидания автобуса).

Задача В. Уборка снега

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вася очень хочет попасть на сборы в СУНЦ и, к счастью, живет недалеко от него. Поэтому он решил дойти до места проведения сборов пешком. Васе известен план города — какие перекрестки соединены улицами и сколько времени требуется, чтобы пройти по каждой улице. Движение по любой улице разрешено в обе стороны. Администрация города, однако, решила устроить в этот день уборку улиц от снега. Если на какой-то из улиц происходит уборка, то движение по ней замедляется в два раза. В распоряжении города есть K снегоуборочных машин. Вася хочет добраться до места назначения как можно быстрее, однако у главы администрации есть с Васей старые счеты, поэтому он хочет максимально замедлить его движение. В результате всякий раз, когда Вася оказывается на перекрестке, глава администрации выбирает не более K улиц, на которых будет производиться уборка, пока Вася перемещается с текущего перекрестка до следующего. Дом Васи находится около перекрестка с номером 1, а СУНЦ — около перекрестка с номером N . Таким образом, перемещение Васи от дома до СУНЦа выглядит следующим образом. В начале глава выбирает дороги, на которых будет проводиться уборка, затем Вася выбирает улицу, по которой он пойдет от перекрестка 1 (Вася достаточно наблюдателен, чтобы заметить, на каких улицах идет уборка). Когда он доходит до конца выбранной улицы и оказывается на перекрестке, процесс повторяется: глава вновь выбирает улицы для уборки, и машины туда мгновенно перемещаются, а затем Вася — улицу, по которой идти, и т. д. Процесс продолжается, пока Вася не попадет в СУНЦ. Ваша задача — выяснить, за какое минимально возможное время Васе удастся достичь СУНЦа при условии, что глава администрации всегда действует оптимально.

Формат входного файла

Первая строка входного файла содержит числа N — количество перекрестков в городе, M — количество улиц и K — количество снегоуборочных машин ($1 \leq N \leq 100, 0 \leq K \leq M \leq 20000$). Следующие M строк содержат описания улиц в следующем формате: a и b — номера перекрестков, которые данная улица соединяет, t — время движения по данной улице (целое положительное число, не превосходящее 1000).

Формат выходного файла

Выведите в выходной файл одно число — минимальное время, за которое Вася может добраться до СУНЦа. или -1 , если добраться туда невозможно.

Пример

stdin	stdout
5 4 2 1 2 1 2 4 1 1 4 1 3 5 1	-1
4 6 2 1 2 1 1 3 1 1 4 2 2 3 1 2 4 1 3 4 1	3

Разбор задачи В. Уборка снега

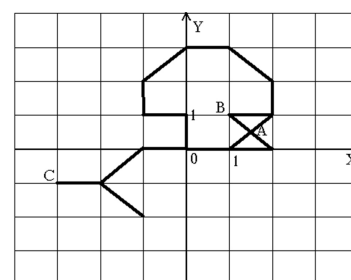
Данную задачу необходимо решать алгоритмом похожим на алгоритм Дейкстры начиная с последней вершины. Будем искать кратчайшее расстояние $D[V]$ с учетом действий администрации города от каждой вершины до конечной вершины N . Изначально такое расстояние известно только для вершины N и равно 0. Выбрав вершину V с наименьшим таким расстоянием, можно пометить вершину V черным цветом и считать, что расстояние до нее подсчитано верно. Необходимо пересчитать расстояние для всех вершин, которые имеют ребро в V . Чтобы посчитать потенциальное расстояние для вершины U необходимо выбрать $K + 1$ ребро (допустимы кратные ребра), с минимальной суммой $D[V] + W(V\text{-черная вершина}, W - \text{вес ребра из } U \text{ в } V)$. Очевидно, что администрация будет удваивать K минимальных ребер. Поэтому результатом для $D[U]$ будет минимум по первым K вершинам $D[V] + 2 * W$ и $D[V] + W$ для $K + 1$ вершины. Если черных вершин, в которые ведут ребра из U , меньше $K + 1$, то некоторые элементы минимума опускаются. Пересчет расстояния для всех вершин после добавления новой черной вершины происходит за время пропорциональное $O(M \log M)$. Так как количество выборов минимума не более N , то общее время работы $O(N * M \log M)$. Если для каждой вершины под-

держивать две кучи со значениями $D[V] + 2 * W$ и $D[V] + W$, то данную задачу можно решить за время $O(M \log M \log K)$.

Задача С. Юный поджигатель

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

На клеточном поле введена система координат так, что центр координат находится в точке пересечения линий сетки и оси направлены вдоль линий сетки. На этом поле выложили связную фигуру, состоящую из спичек. Использовались спички двух типов:



- Спички длины 1 выкладывались по сторонам клеток.
- Спички длины $\sqrt{2}$ выкладывались по диагоналям клеток.

Ребенок хочет сжечь фигуру. При этом он может поджечь ее в одной точке, имеющей целочисленные координаты (например, в точке А на рисунке поджигать фигуру нельзя, а в точках В и С — можно).

Известно, что огонь распространяется вдоль спички равномерно (но по каждой спичке — со своей скоростью). Спичка может гореть в нескольких местах (например, когда она загорается с двух концов; или когда в середине диагональной спички огонь распространяется с одной спички на другую — огонь расползается по вновь подожженной спичке в обе стороны). Напишите программу, которая определит, в какой точке нужно поджечь фигуру, чтобы она сгорела за минимальное время.

Формат входного файла

Во входном файле записано сначала число N — количество спичек ($1 \leq N \leq 40$). Затем идет N пятерок чисел вида X_1, Y_1, X_2, Y_2, T , задающих координаты концов спички и время ее сгорания при условии, что она будет подожжена с одного конца (гарантируется, что каждая спичка имеет длину 1 или $\sqrt{2}$, все спички образуют связную фигуру, и положение никаких двух спичек не совпадает). Все координаты — целые числа, по модулю не превышающие 200, время сгорания — натуральное число, не превышающее 10^7 .

Формат выходного файла

Выведите координаты целочисленной точки, в которой нужно поджечь фигуру, чтобы она сгорела за наименьшее время, а затем время, за которое в этом случае фигура сгорит. Время должно быть выведено с точностью не менее 2-х знаков после десятичной точки. Если решений несколько, выведите любое из них.

Пример

stdin	stdout
1 0 0 1 1 1	0 0 1.00
5 0 0 0 1 1 1 0 0 1 10 0 0 1 0 1 0 0 1 1 1 2 2 1 1 1	0 0 3.25
3 1 1 1 2 10 1 2 2 2 10 1 1 2 2 50	2 2 35.00
16 0 0 0 1 1 -2 -1 -3 -1 1 -2 -1 -1 0 1 -2 -1 -1 -2 1 -1 0 0 0 1 0 3 1 3 1 1 3 2 2 1 2 2 2 1 1 2 1 1 0 1 2 0 1 1 1 2 0 1 0 1 2 1 1 1 1 0 0 1 0 1 0 1 -1 1 1 -1 1 -1 2 1 0 3 -1 2 1	0 0 4.50

Разбор задачи С. Юный поджигатель

Прежде всего заметим, что спички могут пересекаться только концами, либо серединами. Других случаев пересечения быть не может. Таким образом, если разбить каждую спичку на две «половинки» вдвое меньшей длины, то полученные «полуспички» пересекаться будут только концами. Если все координаты исходных спичек умножить на два, то координаты всех «полуспичек» также будут выражаться целыми числами. Далее, если на два умножить также и время горения всех спичек, то время горения «полуспички» также будет выражаться целым числом секунд. Будем считать, что мы так уже поступили, и далее спичками именуется именно такие «полуспички».

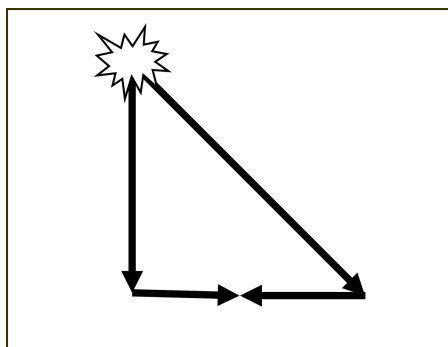
Мы перешли к аналогичной задаче с вдвое большим количеством спичек с целыми координатами, пересекающихся только концами. Построим граф, ребрами которого будут спички, а вершинами – их концы. Задача сводится

к нахождению такой вершины графа, при «поджигании» которой весь граф сгорает за минимальное время.

Будем решать задачу перебором по всем потенциальным «точкам поджигания». Так как в любом случае необходимо в выходной файл вывести кроме координат оптимальной точки общее время сгорания, задачу о нахождении времени сгорания графа при «поджигании» данной вершины тоже надо уметь решать.

Пусть нам дан граф, в котором каждому ребру сопоставлено время сгорания соответствующей ему спички (эту величину будем называть весом или длиной ребра), и в нем зафиксирована некоторая вершина. Как найти время «сгорания» этого графа при «поджигании» этой вершины?

Ясно, что время сгорания графа равно расстоянию от зафиксированной вершины до наиболее удаленной от нее точки графа. Именно «наиболее удаленной точки», а не «наиболее удаленной вершины»! Простейший пример, где эти понятия различаются – треугольник:



Допустим, в приведенном выше примере времени горения вертикальной и диагональной спичек – одна секунда, а время горения горизонтальной спички – четыре секунды. Тогда при поджигании этой фигуры в верхней точке через секунду огонь достигнет обоих концов основания. Еще две секунды потребуется пламени, чтобы сжечь основание. Таким образом, хотя самая удаленная вершина находится на расстоянии 1 от точки поджигания, суммарное время сгорания такой фигуры равно трем секундам.

Вычислим кратчайшие расстояния от данной вершины до всех остальных. Кратчайшее расстояние соответствует моменту времени, когда огонь достигнет данной вершины. Для нахождения расстояний можно использовать, например, алгоритм Флойда.

Алгоритм Флойда находит кратчайшие расстояния между всеми парами вершин в графе, делая число операций, пропорциональное кубу числа вершин графа. Программа, реализующая алгоритм Флойда, выглядит следующим образом:

```
1 for k:=1 to N do
```

```

2  for i:=1 to N do
3    for j:=1 to N do
4      if a[i,j]<$a[i,k]+a[k,j] then a[i,j]:=a[i,k]+a[k,j];

```

Остановимся на описании алгоритма Флойда более подробно. Пусть в матрице $A[i,j]$ записаны длины ребер графа (элемент $A[i,j]$ равен весу ребра, соединяющего вершины с номерами i и j , если же такого ребра нет, то в соответствующем элементе записано некоторое очень большое число, например, 10^9). Построим новые матрицы $C_k[i,j]$ ($k=0, \dots, N$). Элемент матрицы $C_k[i,j]$ будет равен минимальной возможной длине такого пути из i в j , что в качестве промежуточных вершин в этом пути используются вершины с номерами от 1 до k . То есть рассматриваются пути, которые могут проходить через вершины с номерами от 1 до k (а могут и не проходить через какие-то из этих вершин), но заведомо не проходят через вершины с номерами от $k+1$ до N . В матрицу записывается длина кратчайшего из таких путей. Если таких путей не существует, записывается то же большое число, которым обозначается отсутствие ребра.

Сформулируем следующие факты.

В матрице $C_0[i,j]$ записаны длины путей, которые не содержат ни одной промежуточной вершины. Таким образом, матрица $C_0[i,j]$ совпадает с исходной матрицей $A[i,j]$.

В матрице $C_N[i,j]$ записаны минимальные длины путей, которые в качестве промежуточных вершин используют все вершины графа — то есть длины кратчайших путей, которые мы хотим получить.

Если у нас уже вычислена матрица $C_{k-1}[i,j]$, то элементы матрицы $C_k[i,j]$ можно вычислить по следующей формуле: $C_k[i,j] := \min(C_{k-1}[i,j], C_{k-1}[i,k] + C_{k-1}[k,j])$. В самом деле, рассмотрим кратчайший путь из вершины i в вершину j , который в качестве промежуточных вершин использует только вершины с номерами от 1 до k . Тогда возможно два случая:

Этот путь не проходит через вершину с номером k . Тогда его промежуточные вершины — это вершины с номерами от 1 до $k-1$. Но тогда длина этого пути уже вычислена в элементе $C_{k-1}[i,j]$.

Этот путь проходит через вершину с номером k . Но тогда его можно разбить на две части: сначала мы из вершины i доходим оптимальным образом до вершины k , используя в качестве промежуточных вершины с номерами от 1 до $k-1$ (длина такого оптимального пути вычислена в $C_{k-1}[i,k]$), а потом от вершины k идем в вершину j опять же оптимальным способом, и опять же используя в качестве промежуточных вершин только вершины с номерами от 1 до k ($C_{k-1}[k,j]$).

Выбирая из этих двух вариантов минимальный, получаем $C_k[i,j]$.

Последовательно вычисляя матрицы C_0, C_1, C_2 и т.д. мы и получим искомую матрицу C_N кратчайших расстояний между всеми парами вершин

в графе.

Заметим теперь, что при вычислении очередной матрицы C_k нам нужны лишь элементы матрицы C_{k-1} , поэтому можно не хранить в памяти N таких матриц, а обойтись двумя — той, которую мы сейчас вычисляем, и той, которую мы вычислили на предыдущем шаге. На самом деле оказывается, что даже это излишне — все вычисления можно производить в одной матрице (подумайте, почему).

Вернемся к решению нашей задачи.

После нахождения кратчайших путей из «поджигаемой» вершины во все остальные, нам известно время, за которое огонь достигнет каждой из вершин. Теперь нужно проверить все ребра-спички на предмет того, сгорели ли они уже в процессе перемещения огня до вершин, а если нет, то нужно найти время, когда догорит данное ребро. Максимальное из этих времен даст ответ.

Пусть огонь достигает концов спички со временем сгорания L в моменты времени T_1 и T_2 . Если $T_1 = T_2 + L$ или $T_2 = T_1 + L$, то огонь передавался по этой спичке, и максимум из T_1 и T_2 и будет тем временем, к которому спичка сгорит полностью. Отметим, что разность между T_1 и T_2 не может быть больше L (подумайте, почему).

Пусть теперь разность между T_1 и T_2 не равна L . Это значит, что к разным концам спички огонь подошел разными путями, она будет гореть одновременно с обеих сторон и догорит где-то посередине. Напомним, что под спичкой мы понимаем половину спички, и пересекаться не в концах она уже ни с чем не может. То есть поджечь такую спичку никакую другую спичку не может — с обеих ее концов уже и так бушует пламя! В простейшем случае, если спичка подожжена одновременно с обоих концов, она сгорает за время $L/2$. Трудность в том, что в общем случае время возгорания концов спички может не совпадать.

Можно вычесть одно и то же число из T_1 и из T_2 , т.е. мы можем перейти к задаче, где один конец загорается в момент времени 0, а второй — в момент времени T . Общее время сгорания спички в таком случае будет равно $T + (L-T)/2$. Прийти к этой формуле проще всего из следующих соображений. Пусть спичка имеет длину L сантиметров и горит со скоростью 1 сантиметр в секунду. Тогда первые T секунд она будет гореть с одного конца, и сгорит на T см., а оставшееся время потребуется на то, чтобы сжечь $L-T$ см со скоростью 2 см/сек, т.е. время сгорания этого куса спички будет равно $(L-T)/2$. При $T = 0$ формула дает ответ $L/2$, а при $T = L$ — ответ L , что полностью согласуется с условием задачи.

Мы полностью умеем решать задачу о нахождении времени сгорания данной фигуры из спичек при ее поджигании в данной точке. Для этого нужно в соответствующем данной фигуре графе найти максимум из времен

«догораний» каждого из ребер. И не забыть разделить его пополам – ведь при построении графа мы удвоили время сгорания каждой из спичек.

Теперь мы легко можем решить задачу перебором по вершинам. Проверив все потенциальные точки поджога и выбрав из них ту, при поджоге которой время сгорания минимально, мы найдем ответ. Необходимо учесть, что не каждая из вершин построенного графа может быть точкой «поджигания». Так как мы раздвоили каждую спичку, в наш граф войдут также вершины, соответствующие серединам спичек. Из всех точек мы должны выбрать те, координаты которых нацело делятся на 2.

Еще одно замечание – в данной задаче мы всюду работали с целыми числами. Но в двух местах это целое число делилось на два – при нахождении координат пересечения спичек и при вычислении времени сгорания спички, подожженной с обеих сторон. Из этого следует, что общее время сгорания можно представить в виде $X/4$, где X – целое число. Соответственно, дробная часть этого времени будет равна 0.0, 0.25, 0.5 или 0.75, и двух десятичных знаков для ее представления вполне достаточно.

Приведем полный текст программы:

```

1 Program Matches;
2
3 Const
4   TaskID='f';
5   InFile=TaskID+'.in';
6   OutFile=TaskID+'.out';
7
8 Const
9   MaxN=42; { Ограничение на N }
10  MaxG=2*MaxN+1; { Ограничение на число вершин в графе }
11  Infinity=MaxLongInt; { "Бесконечное" расстояние }
12
13 Var
14   N:Integer; {
15   Match:Array[1..MaxN]Of Record { Входные
16     X1,Y1,X2,Y2:Integer; { данные }
17     Time:LongInt; {
18   End; {
19
20   NG:Integer; {
21   Vertex:Array[1..MaxG]Of Record {
22     X,Y:Integer; { Граф }
23   End; {
24   Edge,Distance:Array[1..MaxG,1..MaxG]Of LongInt; {
25
26   Res:Extended; { Минимальное время сгорания }
27   ResX,ResY:Integer; { Оптимальная точка поджога }
28
29 Procedure Load;
30 Var
31   I:Integer;

```

```

32 Begin
33   Assign(Input, InFile);
34   ReSet(Input);
35   Read(N);
36   For I:=1 To N Do
37     With Match[I] Do
38       Read(X1, Y1, X2, Y2, Time);
39   Close(Input);
40 End;
41
42
43
44 Function GetVertex(VX, VY:Integer):Integer;
45   { Функция, возвращающая номер вершины с заданными координатами.
46     При отсутствии нужной вершины она создаётся }
47 Var
48   I:Integer;
49 Begin
50   For I:=1 To NG Do
51     With Vertex[I] Do
52       If (X=VX) And (Y=VY) Then Begin
53         GetVertex:=I;
54         Exit;
55       End;
56
57   Inc(NG); { Если нужная вершина не найдена }
58   With Vertex[NG] Do Begin
59     X:=VX;
60     Y:=VY;
61     For I:=1 To NG-1 Do Begin
62       Edge[I, NG]:=Infinity;
63       Edge[NG, I]:=Infinity;
64     End;
65     Edge[NG, NG]:=0;
66   End;
67   GetVertex:=NG;
68 End;
69
70 Procedure AddEdge(X1, Y1, X2, Y2:Integer; Time:Longint);
71   { Функция, добавляющая ребро между двумя точками }
72 Var
73   A, B:Integer;
74 Begin
75   A:=GetVertex(X1, Y1);
76   B:=GetVertex(X2, Y2);
77   Edge[A, B]:=Time;
78   Edge[B, A]:=Time;
79 End;
80
81 Procedure BuildGraph; { Процедура построения графа }
82 Var
83   I:Integer;
84 Begin
85   NG:=0;

```

```

86  For I:=1 To N Do
87      With Match[I] Do Begin
88          AddEdge(X1*2, Y1*2, X1+X2, Y1+Y2, Time);
89          AddEdge(X1+X2, Y1+Y2, X2*2, Y2*2, Time);
90      End;
91 End;
92
93 Procedure FindShortestPaths;
94 Var
95     K, I, J:Integer;
96 Begin
97     Distance:=Edge;
98     For K:=1 To NG Do
99         For I:=1 To NG Do If Distance[I, K]<Infinity Then
100             For J:=1 To NG Do If Distance[K, J]<Infinity Then
101                 If Distance[I, K]+Distance[K, J]<Distance[I, J] Then
102                     Distance[I, J]:=Distance[I, K]+Distance[K, J];
103 End;
104
105
106
107 Function BurnAt(At:Integer):Extended;
108 { Функция, вычисляющая время сгорания при поджоге в точке At }
109 Var
110     I, J:Integer;
111     Cur, ThisEdge:Extended;
112 Begin
113     Cur:=0;
114     For I:=1 To NG Do If Distance[At, I]>Cur Then Cur:=Distance[At, I];
115     For I:=1 To NG Do
116         For J:=I+1 To NG Do If Edge[I, J]<Infinity Then Begin
117             If (Distance[At, I]<Distance[At, J]+Edge[I, J]) And
118                 (Distance[At, J]<Distance[At, I]+Edge[I, J]) Then Begin
119                 If Distance[At, I]<Distance[At, J] Then
120                     ThisEdge:=Distance[At, J]+(Edge[I, J]-(Distance[At, J]-Distance[At, I]
121                     ))/2
122                 Else
123                     ThisEdge:=Distance[At, I]+(Edge[I, J]-(Distance[At, I]-Distance[At, J]
124                     ))/2;
125                 If ThisEdge>Cur Then Cur:=ThisEdge;
126             End;
127         End;
128     End;
129     BurnAt:=Cur;
130 End;
131
132 Procedure Solve;
133 Var
134     I:Integer;
135     Cur:Extended;
136 Begin
137     Res:=Infinity;
138     For I:=1 To NG Do
139         With Vertex[I] Do
140             If Not Odd(X) And Not Odd(Y) Then Begin

```

```

138         Cur:=BurnAt(I);
139     If Cur<Res Then Begin
140         Res:=Cur;
141         ResX:=X Div 2;
142         ResY:=Y Div 2;
143     End;
144 End;
145 End;
146
147 Procedure Save;
148 Begin
149     Assign(Output,OutFile);
150     ReWrite(Output);
151     WriteLn(ResX, ' ',ResY);
152     WriteLn(Res/2:0:2);
153     Close(Output);
154 End;
155
156 Begin
157     Load;
158     BuildGraph;
159     FindShortestPaths;
160     Solve;
161     Save;
162 End.

```

Задача D. Реклама

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

В супермаркете решили время от времени транслировать рекламу новых товаров. Для того, чтобы составить оптимальное расписание трансляции рекламы, руководство супермаркета провело следующее исследование: в течение дня для каждого покупателя, посетившего супермаркет, было зафиксировано время, когда он пришел в супермаркет, и когда он из него ушел.

Менеджер по рекламе предположил, что такое расписание прихода-ухода покупателей сохранится и в последующие дни. Он хочет составить расписание трансляции рекламных роликов, чтобы каждый покупатель услышал не меньше двух рекламных объявлений. В то же время он выдвинул условие, чтобы два рекламных объявления не транслировались одновременно и, поскольку продавцам все время приходится выслушивать эту рекламу, общее число рекламных объявлений за день было минимальным.

Напишите программу, которая составит такое расписание трансляции рекламных роликов. Рекламные объявления можно начинать транслировать только в целые моменты времени. Считается, что каждое рекламное объявление заканчивается до наступления следующего целого момента времени. Если рекламное объявление транслируется в тот момент времени, когда покупатель входит в супермаркет или уходит из него, покупатель это объявление услышать успевает.

Формат входного файла

Во входном файле записано сначала число N — количество покупателей, посетивших супермаркет за день ($1 \leq N \leq 3000$). Затем идет N пар натуральных чисел A_i, B_i , задающих соответственно время прихода и время ухода покупателей из супермаркета ($0 < A_i < B_i < 10^6$).

Формат выходного файла

В выходной файл выведите сначала количество рекламных объявлений, которое будет протранслировано за день. Затем выведите в возрастающем порядке моменты времени, в которые нужно транслировать рекламные объявления.

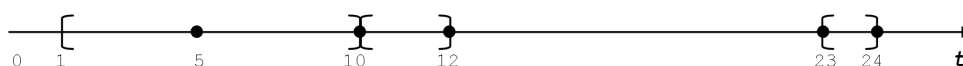
Если решений несколько, выведите любое из них.

Пример

stdin	stdout
5	5
1 10	5 10 12 23 24
10 12	
1 10	
1 10	
23 24	

Разбор задачи D. Реклама

При решении задачи всегда очень полезно представить какую-либо ее визуальную интерпретацию. Отообразим все время работы магазина временной осью, а время прихода и ухода покупателей — отрезками на этой оси. Теперь задачу можно переформулировать: поставить на оси минимальное количество точек с целочисленными координатами так, чтобы в каждом отрезке содержалось не менее двух точек. Пример из условия в такой интерпретации будет выглядеть следующим образом:



В программе будем использовать следующие типы данных:

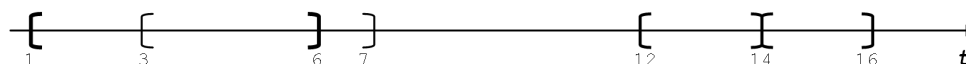
```

1 const
2   MaxN=3000;
3   Infinity=MaxLongInt; {"бесконечная" координата}
4
5 type
6   TSegment = record
7     left, right : longint;
8   end;
9
10 var
11   n : longint;                                {количество отрезков}
12   segment : array [1..MaxN] of TSegment;    {координаты отрезков}
13   point : array [1..MaxN*2] of longint;     {точки, которые мы расставляем}

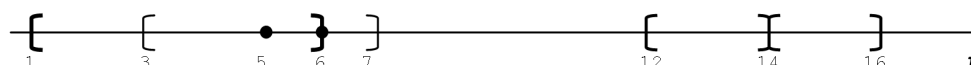
```

Отсортируем все отрезки по возрастанию правых границ, а при их равенстве – по убыванию левых. Ограничение на количество отрезков ($N \leq 3000$) позволяет применить не только быструю сортировку, но и алгоритм сортировки с квадратичной временной сложностью, например метод «пузырька».

Перейдем теперь к основной части решения. Для этого рассмотрим следующий пример. Пусть дано четыре отсортированных отрезка: $[1,6]$, $[3,7]$, $[12,14]$, $[14,16]$.



В самом первом отрезке $[1,6]$ должны содержаться две точки. Их необходимо поставить как можно правее, то есть в точках с целочисленными координатами 5 и 6. Почему? Поскольку это отрезок с наименьшей правой границей, то раньше него другие отрезки не могли закончиться. Но могли начаться другие отрезки! А чем правее мы располагаем точки, тем больше шанс, что они одновременно попадут и в другие отрезки – что нам выгодно. Еще раз обратим внимание на факт, что не существует более выгодной расстановки точек, чем данная: поскольку никакой другой отрезок раньше наших точек не заканчивается, то мы не упускаем ни одну потенциальную возможность улучшить расстановку точек.



В нашем примере расставленные две точки сразу попали и в отрезок $[3,7]$. А вот если бы мы поставили точки, например, в координатах 1 и 2,

то такая расстановка была бы неэффективной – мы покрыли бы ею только один отрезок, а не два. Назовем точку 6 последней расставленной точкой, а точку 5 – предпоследней. В программе это запишется следующим образом:

```
1 PrevLast := right-1;
2 Last := right;
3 {где right – правая граница текущего отрезка.}
```

Переходим к следующему отрезку $[3, 7]$ – но внутри него уже стоят две точки, поскольку левая граница отрезка меньше, чем предпоследняя расставленная точка.

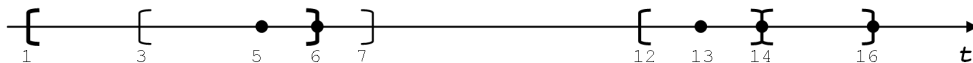
```
1 if left <= PrevLast then <ничего расставлять не нужно>;
```

Следующий отрезок – $[12, 14]$. В нем не стоит еще ни одной точки, так как $left > last$. Из аналогичных соображений ставим в нем две точки как можно правее.



Последний отрезок – $[14, 16]$. В нем уже содержится одна из поставленных точек, так как $Last = left$. Ставим еще одну точку в правую границу отрезка – координата равна 16. При этом предыдущей точкой станет точка 14.

```
1 PrevLast := Last;
2 Last := right;
```



Легко видеть, что такой алгоритм расстановки точек всегда дает оптимальный результат. Итак, для каждого отрезка мы смотрим, нужно ли поставить в нем одну или две точки и если да, то ставим их как можно правее. Резюмируя все вышесказанное приведем ключевой фрагмент программного кода, реализующий данную логику:

```
1 procedure solve;
2 var
3   Last, PrevLast : longint; {две последние поставленные точки}
4   i : longint;
5 begin
6   res := 0; {количество поставленных точек}
7   Last := -Infinity;
8   PrevLast := -Infinity;
9   for i := 1 to N do
10    with segment[i] do
11      if last < left then {необходимо поставить ещё две точки}
12        begin
```



```

13         inc(res);
14         point[res] := right-1;
15         inc(res);
16         point[res] := right;
17         PrevLast := right-1;
18         Last := right;
19     end
20 else
21     if PrevLast < left then    {необходимо поставить ещё одну точку}
22     begin
23         inc(res);
24         point[res] := right;
25         PrevLast := Last;
26         Last := right;
27     end;
28 end;
```

Задача Е. Космический мусорщик

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

В околоземном космическом пространстве накопилось много мусора, поэтому ученые сконструировали специальный аппарат – ловушку для космического мусора. Для того, чтобы хорошо собирать мусор, этот аппарат должен двигаться по достаточно сложной траектории, сжигая собранный по пути мусор. Ловушка может передвигаться в пространстве по 6 направлениям: на север (N), на юг (S), на запад (W), на восток (E), вверх (U) и вниз (D). Движением ловушки управляет процессор. Программа движения задается шестью правилами движения, которые соответствуют каждому из указанных направлений. Каждое такое правило представляет собой строку символов из множества N, S, W, E, U, D. Команда ловушки есть пара из символа направления и параметра – целого положительного числа М. При исполнении такой команды, ловушка в соответствии со своей программой выполняет следующее. Если параметр больше 1, то она перемещается на один метр в направлении, которое указано в команде, а затем последовательно выполняет команды, заданные правилом для данного направления, с параметром меньше на 1. Если же параметр равен 1, то просто перемещается на один метр в указанном направлении. Пусть, например, заданы следующие правила:

Направление	Правило
N	N
S	NUSDDUSE
W	UEWWD
E	
U	U
D	WED

Тогда при выполнении команды S(3) мусорщик выполнит следующие действия:

- 1) переместится на 1 метр в направлении S;
- 2) выполнит последовательно команды N(2), U(2), S(2), D(2), D(2), U(2), S(2), E(2).

Если далее проанализировать действия мусорщика при выполнении команд из пункта 2, получим, что в целом он совершит следующие перемещения: SNNUUSNUSDDUSEDWEDDWEDUUSNUSDDUSEE По заданной команде определите, какое общее количество перемещений на один метр совершит ловушка при выполнении заданной команды. В приведенном примере это количество равно 34.

Формат входного файла

Первые шесть строк входного файла задают правила для команд с направлением N, S, W, E, U и D соответственно. Каждая строка содержит не более 100 символов (и может быть пустой). Следующая строка содержит команду ловушки: сначала символ из множества N, S, W, E, U, D, затем пробел и параметр команды – целое положительное число, не превышающее 100.

Формат выходного файла

Выведите в выходной файл единственное число – количество перемещений, которое совершит ловушка. Гарантируется, что ответ не превышает 10^9 .

Пример

stdin	stdout
N NUSDDUSE UEWWD U WED S 3	34

Разбор задачи Е. Космический мусорщик

Данная задача решается с помощью динамического программирования. Для каждого вида команды необходимо посчитать сколько действий сделает команда вида $L(C)$, где L произвольная буква. Тогда для подсчета $L(C + 1)$ достаточно будет знать только значения для всех букв $R(C)$. Пусть $d[L][C]$ – количество шагов, которое сделает мусорщик при команде $L(C)$. Тогда если правило для буквы L выглядит как $ABC...Z$, то $d[L][C+1]$ равно сумме $d[A][C] + d[B][C] + \dots d[Z][C] + 1$.

Задача F. Shortest Path

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дан взвешенный ориентированный граф и вершина s в нем. Для каждой вершины u найти кратчайший путь от s к u

Формат входного файла

В первой строке находятся три целых числа: n , m и s - номера вершин и граней графа и номер начальной вершины, соответственно ($2 \leq n \leq 2\,000$, $1 \leq m \leq 5\,000$).

Следующие m строки содержат описание ребер. Каждое ребро описывается начальной и конечной вершиной и весом. Вес любого ребра - целое число, не превышающее 10^{15} по модулю. Между одной парой вершин может быть несколько ребер. Могут быть петли.

Формат выходного файла

Вывести n строк - длину кратчайшего пути от s к u для каждой вершины u , либо '*', если такого пути нет или '-', если нет кратчайшего пути от s к u .

Пример

stdin	stdout
6 7 1	0
1 2 10	10
2 3 5	—
1 3 100	—
3 5 7	—
5 4 10	*
4 3 -18	
6 1 -1	

Разбор задачи F. Shortest Path

Для решения этой задачи используется алгоритм Форда-Беллмана, описанный в лекции.

Задача G. Дом улыбок

Вход: stdin
 Выход: stdout
 Ограничение по времени: 3 с
 Ограничение по памяти: 256 Мб

Дом улыбок создан, чтобы повышать настроение. В нем есть n комнат. Некоторые из комнат соединены дверьми. Про каждые две комнаты (с номерами i и j), которые соединены дверью, Петя знает величину c_{ij} — на сколько изменяется у него настроение при переходе из комнаты с номером i в комнату с номером j .

Петя задумался: может ли он поднять себе настроение до бесконечности, двигаясь по какому-нибудь циклу? А если может, то какое минимальное количество комнат ему надо будет проходить за один период цикла?

Формат входного файла

В первой строке записаны два целых положительных числа n и m ($1 \leq n \leq 300, 0 \leq m \leq \frac{n(n-1)}{2}$), где n — количество комнат, а m — количество дверей в доме улыбок. Далее идет описание дверей: m строк по четыре целых числа i, j, c_{ij} и c_{ji} ($1 \leq i \leq n, 1 \leq j \leq n; i \neq j; -10^4 \leq c_{ij} \leq 10^4, -10^4 \leq c_{ji} \leq 10^4$). Гарантируется, что любые две комнаты соединяет не более одной двери. Никакая дверь не соединяет комнату саму с собой.

Формат выходного файла

Выведите минимальное количество комнат, которое необходимо проходить за один проход по циклу, бесконечно повышающему наcтроение, или число 0, если такого цикла не существует.

Примеры

stdin	stdout
4 4 1 2 -10 3 1 3 1 -10 2 4 -10 -1 3 4 0 -3	4

Примечание

Напомним, что циклом называется такая последовательность комнат a_1, a_2, \dots, a_k , что a_1 соединена с a_2 , a_2 соединена с a_3 , \dots , a_{k-1} соединена с a_k , a_k соединена с a_1 . Некоторые элементы последовательности могут совпадать, то есть цикл не обязательно должен быть простым. Количество комнат в цикле считается k , длина последовательности. Заметим, что минимальная возможная длина равна двум.

Разбор задачи G. Дом улыбок

Данная задача так же является задачей на алгоритм Форда-Беллмана. В этой задаче необходимо делать итерации алгоритма из каждой вершины, пока расстояние до стартовой не станет отрицательным. Алгоритм необходимо реализовывать в двух массивах, чтобы контролировать количество ребер в пути. Так же необходимо применить несколько отсечений алгоритма: отсечение отсутствия изменений и отсечение просмотра ребер выходящих из уже помеченных черным вершины. Все операции в алгоритме Форда-Беллмана крайне простые, поэтому программа работает достаточно быстро. Рекомендуется хранить граф в матрице смежности, чтобы обеспечить последовательный доступ к памяти.

Задача Н. Сфера

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

На планете, представляющей собой сферу, есть N городов. Передвигаться по планете можно только днем и за один день можно пройти расстояние не больше D (ночевать вне города на планете не представляется возможным). Учтите, что нельзя, чтобы ночь настигала вас в дороге. Требуется за наименьшее количество дней добраться из одного города в другой.

Формат входного файла

В начале входного файла задано целое число N ($1 \leq N \leq 1000$). Затем заданы два целых числа $S1$ и $S2$ ($1 \leq S1, S2 \leq N$; $S1 \neq S2$) — номера городов, путь между которыми необходимо найти. Затем задано целое число R ($0 < R \leq 10^9$) — радиус планеты. Затем задано целое число D ($0 < D \leq 4 \times 10^9$). Следующие N строк задают расположение городов в виде $G1\ T1\ G2\ T2$, где:

$G1$ — вещественное число, задающее широту ($0 \leq G1 \leq 90$);
 $T1$ — символ широты: 'N' — северная, 'S' — южная;
 $G2$ — вещественное число, задающее долготу ($0 \leq G2 \leq 180$);
 $T2$ — символ долготы: 'E' — восточная, 'W' — западная.

Формат выходного файла

В выходной файл выведите минимальное число дней, необходимое для путешествия, или -1 , если путешествие невозможно.

Пример

stdin	stdout
2 1 2 10000 31416 89 S 13.12 W 89 N 78.8 E	1
3 1 3 10000 15000 89.9 S 13.12 W 0.001 S 63.12 W 89 N 78.8 E	-1

Разбор задачи Н. Сфера

Данная задача решается с помощью алгоритма обхода в ширину. Вершинами графа являются точки на сфере, а ребрами соединены точки, расстояние до которых не превосходит D . Для вычисления расстояния необходимо перевести сферические координаты в декартовы на единичной сфере, после чего посчитать скалярное произведение. Это произведение будет равно косинусу угла между векторами. Расстояние по сфере вычисляется как радиус сферы умноженный на угол между векторами.

Задача I. Домой на электричках

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Одна из команд-участниц олимпиады решила вернуться домой на электричках. При этом ребята хотят попасть домой как можно раньше. К сожалению, не все электрички идут от города, где проводится олимпиада, до станции, на которой живут ребята. И, что еще более обидно, не все электрички, которые идут мимо их станции, останавливаются на ней (равно как вообще, электрички останавливаются далеко не на всех станциях, мимо которых они идут). Все станции на линии пронумерованы числами от 1 до N . При этом станция номер 1 находится в городе, где проводится олимпиада, и в момент времени 0 ребята приходят на станцию. Станция, на которую нужно попасть ребятам, имеет номер E . Напишите программу, которая по данному расписанию движения электричек вычисляет минимальное время, когда ребята могут оказаться дома.

Формат входного файла

Во входном файле записаны сначала числа N ($2 \leq N \leq 100$) и E ($2 \leq E \leq N$). Затем записано число M ($2 \leq M \leq 100$), обозначающее число рейсов электричек. Далее идет описание M рейсов электрички. Описание каждого рейса электрички начинается с числа K_i ($2 \leq K_i \leq N$) — количества станций, на которых она останавливается, а далее следует K_i пар чисел, первое число каждой пары задает номер станции, второе — время (время выражается целым числом из диапазона от 0 до 10^9). Станции внутри одного рейса упорядочены в порядке возрастания времени. В течение одного рейса электричка все время движется в одном направлении — либо от города, либо к городу.

Формат выходного файла

В выходной файл выведите одно число — минимальное время, когда ребята смогут оказаться на своей станции. Если существующими рейсами электричек они добраться не смогут, выведите -1 .

Пример

stdin	stdout
5 3 4 2 1 5 2 10 2 2 10 4 15 4 5 0 4 17 3 20 2 35 3 1 2 3 40 4 45	20

Разбор задачи I. Домой на электричках

Эта задача является классической задачей на алгоритм Дейкстры с динамическим весом ребер. Основное отличие от классического алгоритма Дейкстры заключается в том, что к времени движения на электричке необходимо прибавлять время ожидания ее прихода. Так же нужно не забыть, что некоторые электрички к моменту приезда на станцию уже уехали.

Задача J. Поле чудес

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Для игры в "Поле чудес" используется круглый барабан, разделенный на сектора, и стрелка. В каждом секторе записано некоторое число. В различных секторах может быть записано одно и то же число. Однажды ведущий решил изменить правила игры. Он сам стал вращать барабан и называть игроку (который барабана не видел) все числа подряд в том порядке, в котором на них указывала стрелка в процессе вращения барабана. Получилось так, что барабан сделал целое число оборотов, то есть последний сектор совпал с первым. После этого ведущий задал участнику вопрос: какое наименьшее число секторов может быть на барабане? Напишите программу, отвечающую на этот вопрос.

Формат входного файла

Во входном файле записано сначала число N — количество чисел, которое назвал ведущий ($2 \leq N \leq 30000$). Затем записано N чисел, на которые указывала стрелка в процессе вращения барабана. Первое число всегда совпадает с последним (в конце стрелка указывает на тот же сектор, что и в начале). Числа, записанные в секторах барабана, — натуральные, не превышающие 32000.

Формат выходного файла

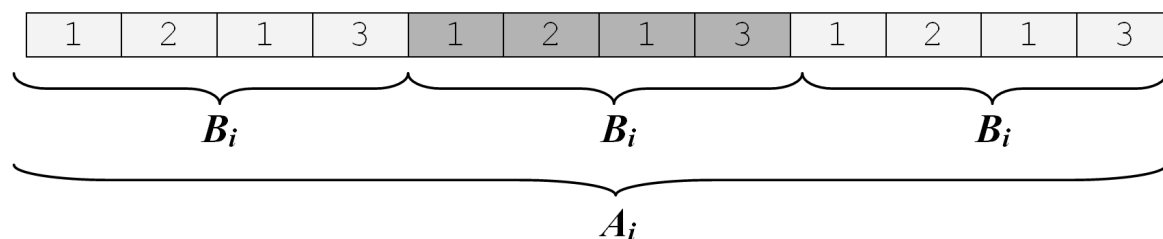
Выведите минимальное число секторов, которое может быть на барабане.

Пример

stdin	stdout
13 5 3 1 3 5 2 5 3 1 3 5 2 5	6
4 1 1 1 1	1
4 1 2 3 1	3

Разбор задачи J. Поле Чудес

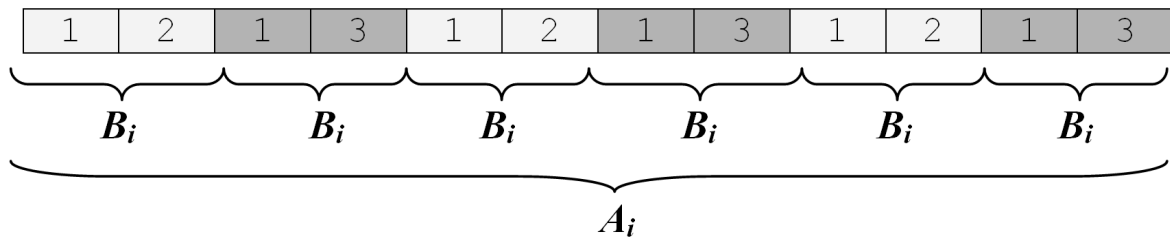
Отбросим последнее число, которое сказал ведущий и сформулируем задачу по другому: дана последовательность чисел A_i длины N . Найти подпоследовательность B_i минимальной длины, повторением которой получена исходная последовательность.



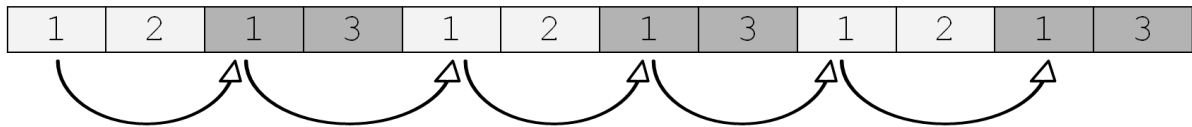
Заметим, что такая последовательность B_i всегда существует: по крайней мере это сама исходная последовательность A_i (повторенная один раз). Ограничения задачи дают возможность решать задачу перебором с некоторыми отсечениями. Пусть последовательность A_i длины N была получена

повторениями последовательности B_i длины K . Поскольку последовательность B_i уложилась в последовательности A_i целое число раз, то N делится на K . Кроме того, K может изменяться в пределах от 1 до N . Будем проверять, является ли K делителем N и если да, образована ли A_i из B_i . Сделать это можно несколькими способами.

Рассмотрим пример. $N = 12$, $K = 2$. Проверяем, образована ли последовательность A_i повторениями из последовательностей длины два. Для этого разобьем A_i на $N/K = 6$ частей и проверим, что получившееся таким образом подпоследовательности B_i одинаковы.



Сначала «по цепочке» проверим, что все первые элементы последовательностей B_i равны.



В программе это будет выглядеть так:

```
1 j := 1;
2 while (j <= n-k) and (a[j] = a[j+k]) do inc(j, k);
```

Затем проверим на равенство все вторые элементы. В нашем примере мы сразу получим неравенство ($2 \neq 3$), поэтому повторением последовательности длины 2 исходная последовательность получена не была.

В общем случае необходимо проверить «по цепочке» на равенство все элементы последовательностей B_i от 1 до K .

Данный алгоритм реализуется следующим образом на языке Pascal:

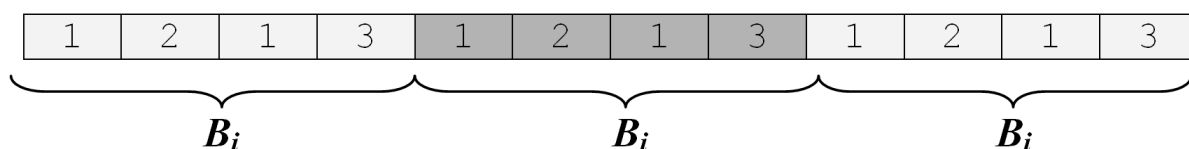
```
1 res := 0;
2 for k := 1 to n do {перебираем все возможные длины в порядке возрастания}
3   if n mod k=0 then
4     begin
5       ok := true;
6       for i := 1 to k do
7         begin
8           j := i;
9           while (j <= n-k) and (a[j] = a[j+k]) do inc(j, k);
```

```

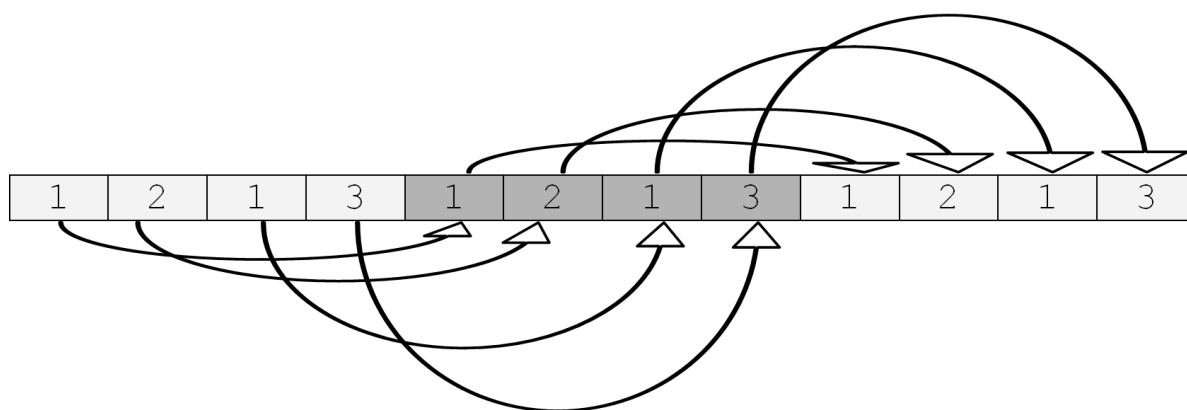
10         if j <= n-k then Ok := false;      { текущая проверка не прошла
11         }
12     end;
13     if Ok then      { все проверки прошли успешно,
14         begin          значит последовательность B найдена }
15         res := k;
16         break;
17     end;
18 end;

```

Можно было поступить по другому. Рассмотрим на примере той же последовательности. $N = 12, K = 4$. Разбиваем A_i на $N/K = 3$ части и опять проверим, совпадают ли все получившееся после такого разбиения подпоследовательности B_i .



Для этого будем последовательно сравнивать на равенство каждый элемент первой подпоследовательности с соответствующим элементом второй, каждый элемент второй последовательности – с соответствующим элементом третьей и так далее. Здесь мы пользуемся свойством транзитивности равенства, то есть свойством, что из равенств $a = b$ и $b = c$ следует, что $a = c$.



Такой способ запрограммировать еще проще. Мы последовательно проходим по последовательности A_i (счетчик j) и каждый раз сравниваем $a[j]$ и $a[j + k]$.

```

1
2 res := 0;
3 for k := 1 to n do {перебираем все возможные длины в порядке возрастания}
4     if n mod k=0 then

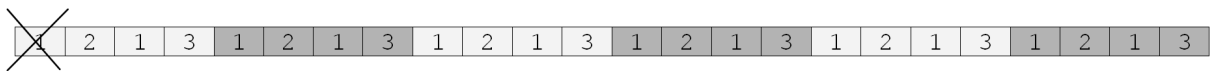
```

```

5      begin
6          j := 1;
7          while (j <= n-k) and (a[j] = a[j+k]) do inc(j);
8          if j > n-k then
9              begin {все проверки прошли успешно, значит последовательность В найдена}
10                 res := k;
11                 break;
12             end;
13         end;

```

Заметим, что отсечение $n \bmod k = 0$ является очень эффективным. Например, число 30000 имеет всего 50 делителей, и мы проверяем 50 длин подпоследовательностей B_i , а не 30000. Однако при больших ограничениях, например $N > 10^6$ переборное решение может уже не успевать находить ответ за заданное время. Для таких случаев задача имеет более красивое решение. Запишем исходную последовательность A_i два раза подряд и выкинем из получившейся последовательности первое число.



В построенной таким образом последовательности будем искать первое вхождение последовательности A_i . Индекс найденного вхождения и будет минимальной длиной последовательности B_i (подумайте, почему это так!) Заметим, что мы всегда найдем вхождение A_i , так как вторая половина построенной последовательности - это A_i .

Таким образом задача свелась к нахождению подстроки в строке. Известны алгоритмы поиска подстроки с линейным относительно длины строки временем исполнения, например алгоритм Кнута-Морриса-Пратта. Подробно об этом алгоритме можно прочитать, например, в книге Т. Кормена, Ч. Лейзерсона, Р. Ривеста «Алгоритмы: построение и анализ», М.: МЦНМО, 2000.

Задача К. Квадрат

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Требуется в каждую клетку квадратной таблицы размером $N \times N$ поставить ноль или единицу так, чтобы в любом квадрате размера $K \times K$ было ровно S единиц.

Формат входного файла

Во входном файле записаны три числа — N , K , S
 $(1 \leq N \leq 100; 1 \leq K \leq N, 0 \leq S \leq K^2)$.

Формат выходного файла

В выходной файл выведите заполненную таблицу. Числа в строке должны разделяться пробелом, каждая строка таблицы должна быть выведена на отдельной строке файла. Если решений несколько, выведите любое из них.

Пример

stdin	stdout
3 2 1	0 0 0 0 1 0 0 0 0
4 2 2	1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0

Разбор задачи К. Квадрат

Это задача на идею. Когда ее знаешь, то решение кажется очевидным. Однако придумать такое решение самому иногда (а точнее даже очень часто) не так-то просто. Раскроем секрет задачи: достаточно сгенерировать любой квадрат $K * K$, содержащий ровно S единиц, а затем просто заполнить им квадрат $N * N$. Теперь сделаем все вышесказанное более аккуратно и подробно. Шаг первый. Необходимо получить любой квадрат размером $K * K$, в котором будет S единиц. Сделаем это каким-либо простым способом. Например, сначала заполним квадрат нулями. Затем будем проходить его по строкам и ставить единицы до тех пор, пока не поставим столько, сколько нам нужно. Если за полный проход по квадрату нам так и не удалось поставить S единиц, то это значит, что задача не имеет решения. Такой случай возможен только при $S > K^2$, а это противоречит условию. Приведем функцию на языке Pascal, которая генерирует необходимый квадрат. Функция *gen* получает размер квадрата k , необходимое число единиц в нем s и массив для записи результата. Функция возвращает *true*, если удалось сгенерировать квадрат, отвечающий нашим требованиям и *false* в противном случае:

```

1 const
2   MaxK=100;
3
4 type
5   Square = array [1..MaxK, 1..MaxK] of byte;
6   {квадрат KxK, повторением которого получаем ответ}
7
8
9
10 function gen(k, s : word; var a : Square) : boolean;
11 var
12   i, j : integer;
13   CurS : word; { сколько единиц уже удалось поставить в квадрате KxK }
14 Begin
15   CurS:=0;
16   fillchar(a, sizeof(a), 0); {вначале заполняем квадрат нулями}
17   for i := 1 to k do
18     for j := 1 to k do
19       if CurS < s then {если число поставленных единиц меньше
20         необходимого}
21         begin
22           a[i, j] := 1; {то ставим очередную единицу}
23           inc(CurS);
24         end
25       end;
26   if CurS < s then gen := false else gen := true;
27 end;

```

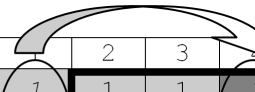
Шаг второй. Распространим полученный квадрат размера $K * K$ на искомый квадрат $N * N$. Сначала рассмотрим, почему это действительно приводит к правильному результату. Пусть $N = 7, K = 3, S = 4$. Приведенная выше функция *gen* получит следующий квадрат:

1	1	1
1	0	0
0	0	0

Назовем его образцом и заполним им квадрат размера $7*7$:

i\j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Обратите внимание, что в заполненном таким образом квадрате любой подквадрат размером 3×3 имеет сумму, равную 4. Почему? Рассмотрим различные подквадраты 3×3 и проследим, что происходит с их суммой. Подквадрат с левым верхним углом (1,1) является образцом, которым мы заполняли большой квадрат, поэтому его сумма, разумеется, равна четырем. Сдвинемся вправо и посмотрим на подквадрат с левым верхним углом в ячейке (1,2)



i \ j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Первый и второй столбец нового квадрата принадлежат как новому, так и старому. А что же приобретенный в результате сдвига новый столбец? Он тоже был в старом, поскольку 4 столбец большого квадрата заполняется тем же самым квадратом-образцом! Выделенный квадрат можно свести к образцу поменяв столбцы – третий на место первого, а первый и второй сдвинуть вправо. А поскольку от перемены мест столбцов сумма элементов квадрата не меняется, то и новый квадрат имеет требуемую сумму. Теперь рассмотрим квадрат, сдвинутый на одну строку вниз. Сдвигаясь вниз, мы целиком «потеряли» первую строчку квадрата-образца, но снова «приобрели» ее! Новый квадрат тоже легко формируется из квадрата-образца, но к перестановке столбцов необходимо добавить перестановку строк.

i \ j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Легко видеть, что как бы мы не сдвигали подквадрат, мы всегда теряем те же самые значения ячеек, что и приобретаем в результате сдвига! Это относится и к случаю, когда мы «упираемся» в границы квадрата $N \times N$.

i\j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Реализовать заполнение квадрата $N \times N$ в программе предельно просто. Приведем ключевой фрагмент программного кода:

```

1 if gen(k, s, a) then      {генерируем образец}
2   for i := 1 to n do     {заполняем образцом квадрат размером N*N}
3     begin                {результат выводим сразу в файл}
4       for j := 1 to n do
5         write(a[(i-1) mod k+1, (j-1) mod k+1], ' ');
6       writeln;
7   end;

```

Задача на самый короткий код

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1 с
 Ограничение по памяти: 64 Мб

Функциональным графом называют ориентированный граф, в котором из каждой вершины выходит ровно одно ребро. Такое название он получил, потому что задает дискретную функцию f на множестве своих вершин. Из каждой вершины x проводится ровно одно ребро в ее образ $f(x)$. Попробуйте максимально короткой программой найти количество компонент сильной связности в этом графе.

Формат входного файла

В первой строке входного файла задано количество вершин в графе. Количество вершин не превосходит 100000. В следующей строке для каждой вершины x указан номер единственной вершины $f(x)$, в которую ведет ребро из вершины x . Вершины нумеруются с единицы.

Формат выходного файла

Выведите единственное число: количество компонент сильной связности заданного графа.

Примеры

stdin	stdout
1 1	1
2 1 2	2
2 2 1	1

Примечание

- Орграф называется сильно связным (англ. *strongly connected*), если любые две его вершины сильно связаны.
- Две вершины s и t любого графа сильно связаны, если существует ориентированный путь из s в t и ориентированный путь из t в s .
- Компонентами сильной связности орграфа называются его максимальные по включению сильно связанные подграфы.

День восьмой (23.02.2013 г.) Контеcт Рипатти Артема Валерьевича

Об авторе...

Рипатти Артем Валерьевич, родился в 1989 году. С начальных классов участвовал в различных олимпиадах. С 9-го класса плотно занялся олимпиадами по математике и физике, с 11-го класса - по информатике. В университете школьные олимпиады сменились соревнованиями по спортивному программированию. Закончил Уфимский государственный авиационный технический университет, сейчас учится в магистратуре.



Основные достижения:

- участие в школьных олимпиадах по математике, физике и информатике зонального и всероссийского уровня.
- дважды финалист чемпионата мира ACM ICPC (2012 и 2013 год).
- финалист чемпионата Russian Code Cup 2012.
- лучший автор задач 2011 года по версии Codeforces.

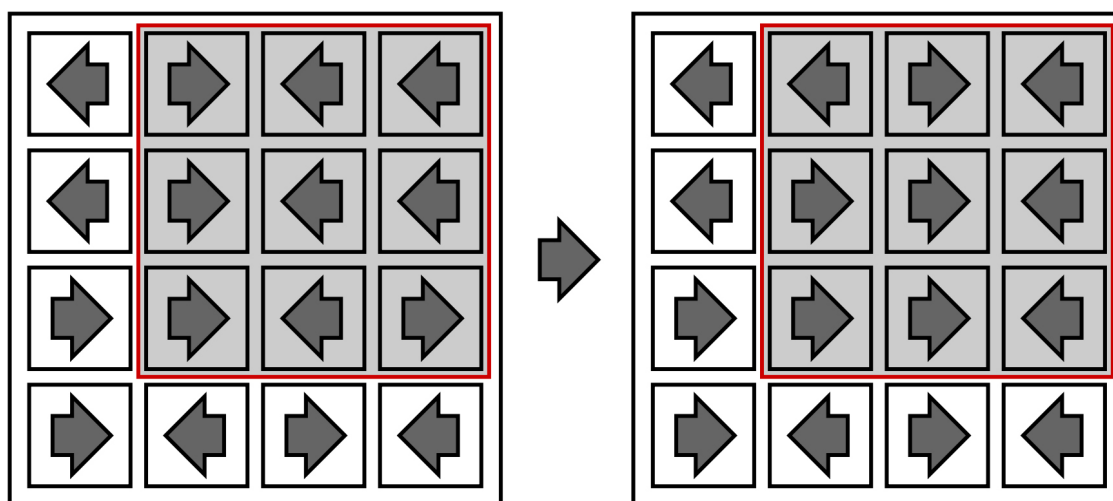
Задачи и разборы

Задача А. Игра со стрелками

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Двое играют в следующую игру.

Имеется коробочка 4×4 с 16 фишками внутри. На каждой из фишек нарисована стрелка, которая указывает налево или направо.



Ход игрока состоит в том, что он выбирает некоторый прямоугольник из фишек так, чтобы стрелки на левой верхней фишке в прямоугольнике и на правой нижней указывали направо. После этого он вытаскивает этот прямоугольник фишек из коробочки (приподнимает их все над коробочкой), поворачивает на 180° и помещает обратно. После этого ход переходит к противнику.

Игрок проигрывает, если не может сделать ход.

Определите, кто из игроков выиграет — тот, кто начинает игру или его оппонент. Игроки играют оптимально. Если игроки, играя оптимально, могут играть бесконечно — объявляется ничья.

Формат входного файла

В первой строке дано число n — количество тестов ($1 \leq n \leq 100$).

Далее идут n тестов, разделенные переводом строк. Каждый тест — 4 строки по 4 символа в каждой — состояние игрового поля в начале игры. Стрелки, указывающие влево обозначены как «<», вправо — «>».

Формат выходного файла

Для каждого теста выведите ответ в отдельной строке в том порядке, в котором тесты даны во входных данных.

Если выигрывает первый игрок — выведите «First», если выигрывает второй игрок — «Second». В случае ничьи выведите «Draw».

Примеры

stdin	stdout
3	Second
<<<<	First
<<<<	Second
<<<<	
<<<<	
<<<<	
<<<<	
<><<	
<<><	
<<<<	
<<>>	
<<>>	
>><<	
>><<	

Разбор задачи А. Игра со стрелками

Представим состояние игрового поля в виде 16-битовой маски, где 0 означает стрелку влево, а 1 — вправо. Можно заметить, что игра ациклическая, поскольку после хода игрока маска строго уменьшается (потому что префикс маски останется тот же самый, а следующая за ним 1 будет заменена на 0).

Дальше задачу следует решать как обычную игру на графе. Будем перебирать маски в порядке их увеличения. После этого будем делать из соответствующего маске состояния все возможные ходы. Если мы достигли ходом хотя бы одно проигрышное состояние — то текущее состояние выигрышное. Иначе текущее состояние проигрышное.

После этого для каждого теста нужно посмотреть выигрышное ли это состояние и вывести «First» или «Second». В этой задаче «Draw» не нужно выводить никогда.

Сложность решения — $O(2^{16} \times 16^3 + n)$.

Задача В. Гарри Поттер и философский камень

Вход: `stdin`
Выход: `stdout`
Ограничение по времени: 1 с
Ограничение по памяти: 256 Мб

Гарри находится в подземном лабиринте. Лабиринт представляет собой прямоугольник, разбитый на $n \times m$ единичных клеток. Некоторые клетки проходимы, а остальные — нет. В одной из клеток расположен выход. Лабиринт со всех сторон окружен непроходимыми стенами, поэтому выход — это единственный способ покинуть лабиринт.

В одной из проходимых клеток находится *философский камень*. Гарри должен найти его и вернуться из лабиринта вместе с ним.

Кроме того, на некоторые проходимые клетки наложены заклинания. *Синий туман* делает проходимую клетку непроходимой для Гарри (философский камень же может там свободно перемещаться). *Красный туман* не позволяет перемещаться по проходимой клетке философскому камню (хотя Гарри может там ходить свободно).

Гарри может делать следующие действия:

- Идти влево, вправо, вверх или вниз, если клетка в данном направлении проходима.
- Если Гарри находится в той же клетке, что и философский камень — Гарри может положить его в карман и далее перемещаться вместе с ним (но при этом теряет способность проходить через клетки с красным туманом).
- Если философский камень у Гарри в кармане — Гарри может его выкинуть. Выкинутый философский камень падает вниз (мгновенно несколько раз перемещается на клетку ниже) до тех пор, пока соседняя снизу клетка не будет для него непроходимой.

Гарри хочет совершить как можно меньше действий и прийти к выходу с философским камнем в кармане. Помогите ему найти оптимальный план действий или скажите, что такого плана нет.

Формат входного файла

В первой строке даны два целых числа n и m ($3 \leq n, m \leq 50$).

В каждой из следующих n строк дано по m символов — описание подземного лабиринта. Возможные символы:

- «.» — проходима клетка.

- «#» — непроходимая клетка (стена).
- «Н» (Harry) — Гарри.
- «Р» (Philosopher's stone) — философский камень.
- «Е» (Exit) — выход.
- «В» (Blue fog) — проходимая клетка с синим туманом.
- «R» (Red fog) — проходимая клетка с красным туманом.

Все клетки на границе прямоугольника непроходимы. Символы «Н», «Р» и «Е» встречаются ровно по одному разу. Под символом «Р» находится символ «#» или «R».

Формат выходного файла

Если выйти из лабиринта с философским камнем невозможно — выведите «IMPOSSIBLE» без кавычек.

Иначе выведите *кратчайшую* последовательность действий для Гарри в виде строки, каждый символ которой кодирует соответствующее действие:

- «L» (Left) — пойти влево.
- «R» (Right) — пойти вправо.
- «U» (Up) — пойти вверх.
- «D» (Down) — пойти вниз.
- «G» (Get) — взять философский камень.
- «T» (Throw) — выкинуть философский камень.

Если возможных ответов несколько — выведите любой.
Смотрите примеры для более ясного понимания.

Примеры

stdin	stdout
<pre> 5 5 ##### #H.P# #B#R# #..E# ##### </pre>	<pre> RRGLLTRRDDLLGRR </pre>
<pre> 4 5 ##### #H#.# #P#E# ##### </pre>	<pre> IMPOSSIBLE </pre>

Разбор задачи В. Гарри Поттер и философский камень

Все, что нам нужно знать в каждый момент времени: положение Гарри, положение философского камня и флаг — камень в кармане у Гарри или нет. Рассмотрим граф, в котором вершинами являются выше рассмотренные состояния, а переходами — возможные действия Гарри. Теперь на этом графе нужно запустить поиск в ширину и найти кратчайшее расстояние до состояния, в котором Гарри находится в клетке выхода, и философский камень — в кармане у Гарри. Далее нужно только восстановить ответ и вывести его. Если никакого пути не нашлось — следует вывести «IMPOSSIBLE».

Сложность решения — $O(n^2m^2)$.

Задача С. Преобразование последовательности

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Имеется последовательность из n целых неотрицательных чисел a_1, a_2, \dots, a_n .

С последовательностью несколько раз делается следующая операция: все элементы последовательности одновременно заменяются в соответствии с формулой $a_i \rightarrow a_i \oplus a_{i+1}$ (для $i = n$ формула выглядит $a_n \rightarrow a_n \oplus a_1$), где \oplus — операция побитового исключающего ИЛИ.

Другими словами, последовательность за одну итерацию превращается из

$$a_1, a_2, \dots, a_n$$

в

$$a_1 \oplus a_2, a_2 \oplus a_3, \dots, a_n \oplus a_1.$$

Ваша задача — определить наименьшее количество применений описанной выше операции к массиву, после которого последовательность будет состоять только из нулей.

Формат входного файла

В первой строке дано число n ($2 \leq n \leq 10^5$).

В следующей строке дано n целых чисел. i -ое из них равно числу a_i ($0 \leq a_i \leq 10^9$).

Формат выходного файла

Выведите одно число — число операций, после которых последовательность будет состоять из одних нулей. Или же выведите «Never» если последовательность не будет иметь требуемый вид после любого числа итераций.

Примеры

stdin	stdout
8 3 6 7 2 4 1 0 5	5
3 0 1 1	Never
5 0 0 0 0 0	0

Примечание

В первом примере последовательность будет изменяться следующим образом:

Изначально: 3, 6, 7, 2, 4, 1, 0, 5;
 Итерация 1: 5, 1, 5, 6, 5, 1, 5, 6;
 Итерация 2: 4, 4, 3, 3, 4, 4, 3, 3;
 Итерация 3: 0, 7, 0, 7, 0, 7, 0, 7;
 Итерация 4: 7, 7, 7, 7, 7, 7, 7, 7;
 Итерация 5: 0, 0, 0, 0, 0, 0, 0, 0.

Разбор задачи С. Преобразование последовательности

Следует заметить, что при совершении 2^k ходов ($k \geq 0$), мы получим последовательность:

$$a_1 \oplus a_{1+2^k}, a_2 \oplus a_{2+2^k}, \dots, a_n \oplus a_{n+2^k}.$$

(здесь и далее, если индекс не попадает в диапазон от 1 до n , то он берется по модулю так, чтобы попадать в этот диапазон).

Рассмотрим случай $n = 2^k$. Ровно через n операций (согласно предыдущему равенству) мы получим последовательность из нулей. Значит искомым ответ находится в пределах от 0 до n .

Пусть теперь $n = 2^m t$ (где t не делится на 2). Рассмотрим сравнение $2^x \equiv 2^m \pmod{2^m t}$. Его можно упростить: $2^{x-m} \equiv 1 \pmod{t}$. По теореме Эйлера оно имеет решения вида $x = m + z\phi(t)$. Для каждого x такого вида последовательность будет иметь вид:

$$a_1 \oplus a_{1+2^m}, a_2 \oplus a_{2+2^m}, \dots, a_n \oplus a_{n+2^m}.$$

Из этого следует, что если какие то два числа a_i и a_{i+2^m} не равны, то последовательности из нулей мы не получим никогда (это легко доказать от противного) и следует выводить «Never». Если же для любого i $a_i = a_{i+2^m}$, то решение задачи эквивалентно решению этой же задачи для $n = 2^m$. Отсюда — и для этого случая ответ находится в диапазоне от 0 до n .

Итак, мы знаем, что если решение есть, то ответ лежит в диапазоне от 0 до n . Точное его значение можно найти при помощи бинарного поиска.

Сложность решения — $O(n \log n)$.

Задача D. Отрезки

Вход:	stdin
Выход:	stdout
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Имеется лист бумаги в клеточку. Лист в клеточку представляет собой множество взаимно перпендикулярных прямых, которые идут вдоль сторон клеточек. Некоторые отрезки на этих прямых могут быть покрашены, а остальные будут не покрашены. Изначально лист чист.

На нем с помощью карандаша в некоторые моменты времени рисуются горизонтальные и вертикальные отрезки. Отрезки идут вдоль сторон клеточек и соединяют какие либо два узла клетчатой сетки. Операция рисования отрезка делает покрашенными соответствующие участки соответствующей

прямой вне зависимости от того, были они до этого покрашены или нет. Рисуемые отрезки могут пересекаться, касаться или накладываться друг на друга.

В некоторые моменты времени некоторые уже нарисованные линии стираются при помощи ластика. Каждое стирание представляет собой горизонтальный или вертикальный отрезок, на котором удаляются все нарисованные линии. Эти отрезки также идут вдоль сторон клеточек и соединяют какие либо два узла клетчатой сетки. То есть операция стирания делает не покрашенными соответствующие участки соответствующей прямой вне зависимости от того, были они до этого покрашены или нет. При этом другие линии, перпендикулярные отрезку стирания, никоим образом не затрагиваются.

При выполнении всех операций вопрос о покраске граничных точек отрезков (все эти точки являются узлами сетки) решается следующим образом. Если после выполнения операции в сколь угодно малой окрестности данной точки имеется хотя бы одна покрашенная точка — то и данная точка должна быть покрашенной. Иначе — данная точка должна быть не покрашенной.

В каждый момент времени можно видеть картину из множества линий, некоторые из которых пересекаются. Случаи касания двух перпендикулярных линий тоже считаются за пересечение. Другими словами, пересечение — это узел клетчатой решетки, из которого выходят хотя бы одна пара нарисованных взаимно перпендикулярных линий.

Ваша задача — находить общее количество пересечений после каждой из операций.

Формат входного файла

В первой строке дано число n — общее число операций ($1 \leq n \leq 10^5$).

Далее идут n строк. В i -ой из них описана операция в виде пятерки « $op\ x_1\ y_1\ x_2\ y_2$ ». op — это тип операции, «add» или «del» для добавления и удаления соответственно. x_1 и y_1 — координаты начала отрезка, x_2 и y_2 — координаты конца отрезка. Гарантируется, что каждый отрезок строго горизонтальный или строго вертикальный ($x_1 = x_2$ или $y_1 = y_2$), а также невырожден ($x_1 \neq x_2$ или $y_1 \neq y_2$).

Все координаты целые от 1 до 10^9 .

Формат выходного файла

Выведите n строк. В i -ой из них должно быть число пересечений на картинке после выполнения i -ой операции.

Примеры

stdin	stdout
7	0
add 3 1 3 6	1
add 2 2 6 2	2
add 5 1 5 6	4
add 1 4 7 4	3
del 5 3 5 5	5
add 3 5 5 5	6
add 5 2 5 7	

Разбор задачи D. Отрезки

Сожмем координаты. Для каждой координаты будем поддерживать множество не пересекающихся отрезков, лежащих на данной координате. Например, мы зафиксировали координату x_0 по оси x . В соответствующем множестве будут находиться все отрезки, у которых x -координаты обоих концов равны x_0 .

Когда поступают запросы на создание отрезков, пересечения нового отрезка со старыми будем отслеживать в соответствующем множестве. Все отрезки, полностью или частично покрытые новым отрезком, следует удалить. После этого мы создаем один отрезок (возможно, он будет «продолжен» теми отрезками, что мы покрыли частично).

При удалении отрезков мы удаляем полностью покрытые отрезки. Частично покрытые отрезки сначала удаляем, а затем добавляем кусочек, который был не покрыт.

Таким образом, изначальный набор запросов будет преобразован в другой набор запросов, где добавляемые отрезки не пересекаются, а удаляем мы каждый раз ранее созданный отрезок полностью. В новом наборе запросов будет порядка $O(n)$, поскольку для каждого старого запроса создается не более 2 отрезков, а каждый созданный отрезок удаляется не более одного раза. Структура данных, вроде `set` из C++, позволяет сделать все преобразование за $O(n \log n)$.

Для ответов на вопросы заведем 2 двумерных дерева отрезков с операциями: изменение в точке, сумма на прямоугольнике (например, подойдет дерево Фенвика). В первом из них мы будем добавлять горизонтальные отрезки и отвечать на запросы для вертикальных отрезков, а во втором — добавлять вертикальные и отвечать на запросы для горизонтальных. Рассмотрим только первое из них — для второй операции будут выполняться аналогично.

При добавлении горизонтального отрезка $(x, y_1) - (x, y_2)$ ($y_1 < y_2$), мы делаем в точке (x, y_1) «+1», а в точке $(x, y_2 + 1)$ — «-1». При запросе для отрезка $(x_1, y) - (x_2, y)$ ($x_1 < x_2$) нам нужно узнать сумму на прямоугольнике с углами в $(x_1, 0)$ и (x_2, y) .

Теперь для решения задачи нужно просто завести глобальную переменную, в которой будет храниться текущий ответ. После этого выполнить новый список запросов, изменяя глобальную переменную и время от времени (когда запрос из изначального списка завершен) выводить ее в выходной файл.

Сложность решения — $O(n \log n)$.

Задача Е. Четыре подмножества

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дано 100 случайных попарно различных чисел из промежутка от 1 до 10^5 . Ваша задача — выбрать из них 4 непустых попарно не пересекающихся подмножества так, чтобы сумма чисел в каждом из них была одна и та же.

Числа генерируются при помощи линейного конгруэнтного генератора псевдослучайных чисел и равномерно распределены на отрезке от 1 до 10^5 .

Известно, что для каждого теста выбрать нужным способом 4 подмножества возможно.

Формат входного файла

Дано 10 строк по 10 чисел в каждой. Каждое число от 1 до 10^5 . Все числа попарно различны.

Формат выходного файла

Выведите 4 строки. В i -ой из них следует вывести числа из i -го подмножества в следующем формате. Сначала идет количество чисел в подмножестве, а затем — сами числа, разделенные пробелом. Смотрите пример для более точного понимания.

Если возможных ответов несколько — выведите любое.

Примеры

stdin	stdout
1 2 3 4 5 6 7 8 9 10	1 100
11 12 13 14 15 16 17	3 20 30 50
18 19 20	5 1 2 3 4 90
21 22 23 24 25 26 27	2 35 65
28 29 30	
31 32 33 34 35 36 37	
38 39 40	
41 42 43 44 45 46 47	
48 49 50	
51 52 53 54 55 56 57	
58 59 60	
61 62 63 64 65 66 67	
68 69 70	
71 72 73 74 75 76 77	
78 79 80	
81 82 83 84 85 86 87	
88 89 90	
91 92 93 94 95 96 97	
98 99 100	

Примечание

Пример не соответствует условию задачи, поскольку сгенерирован не случайным образом — он дан лишь для ознакомления с форматом ввода-вывода.

Разбор задачи Е. Четыре подмножества

У жюри есть решение, для которого нет четкого доказательства корректности. Однако это решение с успехом нашло ответ на 100000 случайных тестов во всех случаях. Суть этого решения в следующем.

Сначала будем искать две пары чисел, в которых сумма одинакова. Найденную четверку чисел исключим из множества и повторим процесс. Как только пары чисел с одинаковой суммой закончатся — будем искать две тройки с одинаковой суммой и исключать уже по 6 чисел из множества. После этого то же самое сделаем с четверками чисел.

В итоге получим k пар групп чисел (k в среднем равно 19-20) $(A_1, B_1), (A_2, B_2), \dots, (A_k, B_k)$, где $A_i = (a_{i_1}, a_{i_2}, \dots)$, а $B_i = (b_{i_1}, b_{i_2}, \dots)$,

причем a -шки и b -шки — это какие то числа, которые даны в самом начале. Понятно, что $\sum_j a_{ij} = \sum_j b_{ij}$ для каждого i . Рассмотрим суммы чисел $\sum_j a_{ij}$. Выпишем эти числа. Теперь будем перебирать все подмножества из данных чисел и находить их сумму. С высокой вероятностью сумма в каких то двух из этим подмножеств совпадет.

Рассмотрим два подмножества X и Y , в которых сумма совпала. Если они пересекаются — исключим из каждого из них их пересечение. Получим два подмножества X' и Y' . Вернемся к парам групп и рассмотрим те, которые заданы первым из найденных подмножеств. То есть те A_i , для которых $i \in X'$. Объединения этих A_i и будет первым подмножеством чисел, которые нужно вывести в ответ. Второе подмножество ответа — объединение B_i таких, что $i \in X'$. Для третьего и четвертого подмножеств X' следует заменить на Y' .

День девятый (24.02.2013 г.)

Контест Копелиовича Сергея Владимировича

Об авторе...

Копелиович Сергей Владимирович родился в Санкт-Петербурге в 1989 году. Закончил физ-мат лицей №30. В 4-6 классах занимался в кружке по математике. Олимпиадами по программированию начал активно заниматься в 9-м классе.

В июне 2012-го закончил Санкт-Петербургский государственный университет. Член научного комитета Всероссийской школьной олимпиады по информатике и сборов по подготовке к международной олимпиаде школьников.

С 2008-2009 учебного года регулярно читает лекции в СПбГУ. Ссылки на планы лекций можно посмотреть здесь <http://kruzhok.spbgu.ru/09e> и здесь <http://195.19.228.2/%7Esk1/mm/345/>

На протяжении года работал стажером в Санкт-Петербургском офисе Яндекса.

С марта 2012-го сотрудник ВКонтакте.

Постоянный участник сборов в Харькове (с 2011-го) и сборов в Петрозаводске (с 2004-го, как участник)



Основные достижения:

- Золотые медали на IOI в 2005 (Польша) и 2006 (Мексика) годах.
- TopCoder Open Algorithm Final 2008, 5-th place.
- Петрозаводские летние сборы 2008 (тренировочные студенческие все-российские) – 1-е командное место.
- Золотой медалист Чемпионата мира по программированию ACM ICPC 2007 года.
- В ACM соревнованиях выступал в команде Burunduchki. Бронзовая и золотая медали в 2008 (Канада) и 2009 (Швеция) годах.

- За последний год: Финалист Google Code Jam и Topcoder Open.

Теоретический материал. Динамика и жадность 20 лет спустя

Часть 1. Проверка деревьев на изоморфность

Определение 1.

Пусть дерево задается матрицей смежности. Два дерева с матрицами смежности c_1 и c_2 называются изоморфными, если существует такая перестановка p , что $\forall i, j : c_1[p_i, p_j] = c_2[i, j]$.

Определение 2.

Изоморфизм корневых деревьев определяется аналогично. Разница в том, что корень переходит в корень ($p_{root_1} = root_2$).

Алгоритм 1. Корневое дерево. $O(n \log n)$.

Даны два корневых дерева. Нужно проверить, изоморфны ли они. Определим хеш вершины, как хеш отсортированного вектора хешей детей. Деревья изоморфны, если хеши корней совпадают. При этом хеш функция должна обладать следующим свойством: хеши различных векторов различны.

Алгоритм 1. Подробности.

Алгоритм работает за $O(n \log n)$, при этом все, кроме сортировки работает за $O(n)$. Важно обратить внимание на выбор хеш функции. В данном случае полиномиальный хеш не подходит. Пример: пусть есть дерево $(1, 2), (1, 3), (1, 4)$ и дерево $(1, 2), (1, 3), (3, 4), (3, 5)$. Хеш от вектора

$$a_0, a_1, \dots, a_k = \sum_{i=0}^k a_i p^i$$

Деревья различны. Тем не менее, хеши от них равны: $1 + 1 \cdot p + 1 \cdot p^2 = 1 + (1 + 1 \cdot p) \cdot p$. Подходит, например, следующая хеш функция:

$$13 + \sum_{i=0}^k \log a_i$$

При реализации на C++ считать ее следует в типе `long double`. Может показаться, что, поскольку функция не зависит от порядка a_i , можно отказаться от сортировки. Это не так. Появится погрешность, поэтому хеш

зависит от порядка. Другой пример хорошей хеш функции:

$$\left(\sum_{i=0}^k a_i^2 + a_i p^i + 3 \right) \mod 2^{64}$$

Алгоритм 2. Произвольное дерево. $O(n \log n)$.

Чтобы приспособить предыдущий алгоритм для обычного некорневого дерева, можно подвесить его за центр. Если центром дерева является ребро, то можно разделить ребро-центр новой вершиной на два ребра, таким образом, центром дерева будет только что добавленная вершина. Деревья изоморфны, если хеши центров совпадают.

Алгоритм 3. Избавляемся от сортировки. $O(n)$.

Давайте для каждой вершины считать ее тип — целое число от 1 до n . Все листья будут иметь тип 1. Чтобы посчитать тип новой вершины, возьмем типы ее детей, отсортируем, посчитаем хеш от получившегося вектора и спросим у хеш-таблицы, какой тип соответствует этому хешу. Если такого хеша еще не было, хеш-таблица вернет первый не использованный тип.

Теперь можно избавиться от сортировки. Будем генерировать «вектор типов детей» автоматически в порядке возрастания:

```

1 list[1] <-- все листья обоих деревьев
2 for type = 1..n
3   for v in list[type]
4     p = parent[v]
5     children[p].push_back(type)
6     if children[p].size() = degree[p] then
7       h = hash(children[p])
8       t[p] = get_type(h)
9       list[t[p]].push_back(parent[v])

```

Здесь мы перебираем вершины в порядке возрастания типа и добавляем наш тип в вектор отца. Деревья изоморфны, если алгоритм, запущенный от всех листьев обоих деревьев выдал, что их корни (для корневых) или центры (для произвольных) имеют одинаковый тип.

Алгоритм 4. Бор вместо хешей и хеш-таблицы.

Зачем нам хеши и хеш-таблица? Чтобы по вектору из чисел от 1 до n получить число от 1 до n . Данную задачу можно решать бором. Для этого нужно вектор (строку) положить в бор и в вершине, в которой кончается вектор, хранить число от 1 до n — тип, соответствующий этому вектору. Тем не менее, так как алфавит имеет размер n , совсем от хеш-таблицы мы

пока не избавились. Чтобы уметь и спускаться по бору за $O(1)$, и быстро добавлять в бор новые вектора, нужно ребра, исходящие из вершины, хранить в хеш-таблице.

Алгоритм 5. Полное $O(n)$. Без структур данных.

Сейчас мы спускаемся по бору каждый раз, когда вектор уже готов. От корня до вершины-конца. Давайте вместо этого спускаться по бору постепенно, пока вектор растет, и хранить для вершины ее текущее положение в боре. То есть, вместо

```
children[p].push_back(type)
```

нужен следующий код,двигающий позицию отца в боре

```
pos[p] = go_down(pos[p], type)
```

Теперь, поскольку `type` \uparrow , для каждой вершины бора нужно хранить только последнее исходящее ребро (ребро с максимальным `type`).

Часть 2. Метод двух указателей для offline задач

Постановка задачи

Дан массив a , и q запросов на отрезке, не меняющих массив. Мы хотим быстро в Offline обработать все запросы. Пример запроса: «сколько различных чисел на отрезке?»

Случай $l_i \uparrow$ и $r_i \uparrow$

Если отрезки можно упорядочить так, что левые и правые концы не убывают ($l_i \leq l_{i+1}, r_i \leq r_{i+1}$), то достаточно научиться обрабатывать две операции — $L++$ и $R++$. Решение задачи «сколько различных чисел на отрезке?»:

```
1 map<int, int> cnt;
2 int answer = 0, L = 0, R = 0;
3 for (int i = 0; i < q; i++) {
4     while (R <= r[i])
5         if (cnt[a[R++]]++ == 0)
6             answer++;
7     while (L < l[i])
8         if (--cnt[a[L++]] == 0)
9             answer--;
10    result[i] = answer;
11 }
```

Время работы примера $O(q \log n)$. Время работы в общем случае $2q \cdot t(n)$, где $t(n)$ — максимальное время обработки одной операции $L++$ или $R++$.

Случай произвольных l_i и r_i

Давайте разделим все запросы на \sqrt{n} групп. i -й запрос попадает в группу номер $\lfloor \frac{l_i}{\sqrt{n}} \rfloor$. Теперь каждую группу отсортируем по r_i и обработаем отдельно. При этом в процессе обработки одной группы указатель R будет только возрастать, а указатель L при переходе к следующему отрезку может как возрастать, так и убывать, но поменяется не более чем на \sqrt{n} .

Оценим время работы: указатель R в сумме сдвинется не более чем на $n\sqrt{n}$ (\sqrt{n} групп, в каждой на n), указатель L в сумме сдвинется не более чем на $q\sqrt{n}$ (для обработки каждого из q запросов произойдет сдвиг не более \sqrt{n}). Получаем суммарное время работы $(q + n)\sqrt{n} \cdot t(n)$, где $t(n)$ — максимальное время обработки одной операции $l--$, $l++$ или $r++$.

Часть 3. Линейная память в квадратных динамиках

Пример задачи, решение

Есть матрица $n \times m$ из натуральных чисел, нужно дойти из $(1, 1)$ в (n, m) , сдвигаться можно только на $(0, 1)$ или на $(1, 0)$. При этом нужно максимизировать сумму чисел на пути. Стандартным решением данной задачи является следующая динамика, работающая за $O(nm)$:

```
1 int f[n + 1][m + 1];
2 f[][] <--- 0
3 for (int i = 1; i <= n; i++)
4     for (int j = 1; j <= m; j++)
5         f[i][j] += a[i][j]
6         f[i + 1][j] = max(f[i + 1][j], f[i][j])
7         f[i][j + 1] = max(f[i][j + 1], f[i][j])
```

Ответ (максимальная сумма) содержится в ячейке $f[n][m]$.

Делаем память линейной

Заметим, что из i -й строки матрицы f мы переходим только в $i + 1$ -ю (следующий слой). Поэтому код можно изменить следующим образом:

```
1 int cc = 0, f[2][m + 1];
2 f[cc] <--- 0
3 for (int i = 1; i <= n; i++)
4     f[cc ^ 1] <-- 0
5     for (int j = 1; j <= m; j++)
6         f[cc][j] += a[i][j]
7         f[cc ^ 1][j] = max(f[cc ^ 1][j], f[cc][j])
8         f[cc][j + 1] = max(f[cc][j + 1], f[cc][j])
9     cc ^= 1
```

Ответ (максимальная сумма) содержится в ячейке $f[cc1][m]$.

Восстановление пути

Чтобы восстановить путь, на котором достигается максимальная сумма, первую версию кода (с квадратной памятью) можно модифицировать, добавив массив $p[n+1][m+1]$, хранящий для каждой точки (i, j) направление, по которому мы в нее пришли. Вторую динамику таким образом не модифицировать. Для восстановления пути нам все еще нужна квадратная память.

Восстановление пути с линейной памятью

Рассмотрим три строки: $i=1$, $i=n/2$, $i=n$. Давайте по ходу динамики насчитаем такой одномерный массив $p[m+1]$, что оптимальный путь из $(1, 1)$ в (n, x) проходит через точку $(n/2, p[x])$. Теперь мы знаем, что оптимальный путь устроен так: $(1, 1) \rightarrow (n/2, p[m]) \rightarrow (n, m)$. Чтобы восстановить путь полностью, нужно реализовать нашу динамику, как рекурсивную функцию, которая ищет путь из (y_1, x_1) в (y_2, x_2) за время $O((y_2 - y_1 + 1) \cdot (x_2 - x_1 + 1))$, и вызвать ее от пары точек $(1, 1)$ и $(n/2, p[m])$ (первая половина пути), а затем от $(n/2, p[m])$ и (n, m) (вторая половина пути). Грубо оценим время работы динамики: $T(n, m) = nm + T(n/2, x) + T(n/2, m - x) = nm + \frac{n}{2}m + T(n/4, x_1) + T(n/4, x - x_1) + T(n/4, x_2) + T(n/4, (m - x) - x_2) = \dots = nm + \frac{n}{2}m + \frac{n}{4}m + \dots \leq 2nm$. Здесь я пользуюсь тем, что на каждом уровне сумма x -ов будет равна m . Грубость оценки заключается в том, что для упрощения технической сложности оценки здесь намерено опущена «плюс единица».

Применяем обретенное знание к задаче о рюкзаке

Постановка задачи: дан рюкзак вместимости s и n вещей со стоимостями c_i и размерами w_i . Каждую вещь можно или целиком положить в рюкзак, или целиком не взять. Нужно максимизировать суммарную стоимость вещей в рюкзаке. Стандартным решением данной задачи является следующая динамика, работающая за $O(ns)$:

```

1 int f[n + 1][s + 1];
2 f[][] <--- 0
3 for (int i = 0; i < n; i++)
4   for (int j = 0; j <= s; j++)
5     f[i + 1][j] = max(f[i + 1][j], f[i][j])
6     if (j + w[i] <= s[i])
7       f[i + 1][j + w[i]] = max(f[i + 1][j + w[i]], f[i][j] + c[i])

```

Отличие этой динамики от предыдущей в том, что переходы теперь не $(1, 0)$ и $(0, 1)$, а $(1, 0)$ и $(1, w[i])$. Мы посчитали оптимальную стоимость. Чтобы получить не только стоимость вещей в рюкзаке, но и сам набор, нам

нужна квадратная память¹, но можно применить такую же оптимизацию, как и выше. Точка $(n/2, p[x])$ будет соответствовать тому, что первые $n/2$ вещей должны занять ровно $p[x]$ места в рюкзаке, соответственно, динамика распадается на две независимых — разместить первые $n/2$ вещей в рюкзаке размера $p[x]$, и разместить оставшиеся $n - n/2$ вещей в рюкзаке размера $s - p[x]$.

Часть 4. Жадность

Сортируем и радуемся

Для начала рассмотрим несколько простых задач, которые решаются сортировкой + циклом `for`.

1. **Условие.** Есть заказы, у каждого заказа есть время выполнения t_i . Заказы можно выполнять в любом порядке. Мы начинаем в момент времени 0 и к моменту времени d должны выполнить максимально возможное количество заказов.

Решение. Выполняем заказы в порядке возрастания t_i .

2. **Условие.** Есть заказы, у каждого заказа есть время выполнения t_i и свой *deadline* d_i . Заказы можно выполнять в любом порядке. Мы начинаем в момент времени 0 и должны успеть выполнить **все** заказы до их *deadline*-ов.

Решение. Выполняем заказы в порядке возрастания d_i .

3. **Условие.** Есть коробки, у каждой коробки есть масса m_i и максимальная суммарная масса, которую можно поставить на коробку сверху, чтобы коробка не сломалась w_i . Нужно построить вертикальную башню, используя **все** коробки.

Решение. Башня снизу вверх — коробки в порядке убывания $m_i + w_i$. Доказательство: самая нижняя коробка обладает свойством $w_i \geq \sum_{j \neq i} m_j \Rightarrow w_i + m_i \geq \sum_j m_j = \text{const} \Rightarrow$ вниз можно смело ставить коробку с максимальным $w_i + m_i$.

4. **Условие.** Есть заказы, у каждого заказа есть время выполнения t_i и штраф w_i . Если i -й заказ начать выполнять в момент времени C_i , то к суммарному штрафу прибавится величина $w_i C_i$. Заказы можно

¹Для задачи о рюкзаке без стоимостей (например, набрать вещей суммарного размера ровно s) можно восстановить ответ без извращений, используя также линейную память. Тем не менее, в нашем случае без извращений не получится.

выполнять в любом порядке. Мы начинаем в момент времени 0. Нужно выполнить все заказы, минимизируя суммарный штраф.

Решение. Выполняем заказы в порядке убывания $\frac{w_i}{t_i}$. Доказательство: рассмотрим два соседних (в порядке выполнения) заказа, выпишем условие, когда их выгодно поменять местами. Условие: $w_2 t_1 > w_1 t_2$.

5. **Условие.** Даны n строк, нужно их сконкатенировать в таком порядке, чтобы конкатенация была лексикографически минимальна.

Решение. Сортируем строки с помощью следующей функции сравнения

```
bool less(string a, string b) { return a + b < b + a; }
```

Доказательство: см. предыдущую задачу.

Сформулируем общий принцип жадного решения задач сортировкой. Нужно придумать правильный компаратор. Чтобы его придумать, достаточно рассмотреть случай $n = 2$ и понять, какой из порядков $(1, 2)$ или $(2, 1)$ выгодней. Один из способов понять, какой из порядков выгодней — ввести функцию оценки. Для задачи (4) использовалась функция оценки $F(1, 2) = w_2 t_1$ «суммарный штраф, который мы заплатим». Для задачи (5) использовалась функция оценки $F(s, t) = st$ «конкатенация». Для произвольной функции F компаратор выглядеть так:

```
bool less(type a, type b) { return F(a, b) < F(b, a); }
```

Заметим также, что, если отношение получилось нетранзитивным, решение заведомо некорректно.

Еще две задачи на сортировку

Решим две более сложные задачи на тему «сортировка».

1. **Условие.** Задача «Два конвейера»: есть n деталей и два конвейера, i -ю деталь нужно обрабатывать сперва a_i единиц времени на первом конвейере, затем b_i на втором. Порядки выполнения деталей на первом и втором конвейерах должны быть одинаковы. Нужно выбрать порядок обработки деталей так, чтобы минимизировать момент времени, когда все детали обработаны.

Решение. Решаем задачу для $n = 2$. $F(1, 2) = a_1 + \max(a_2, b_1) + b_2$. Используем полученную функцию для создания компаратора:

$$i < j \Leftrightarrow a_i + \max(a_j, b_i) + b_j < a_j + \max(a_i, b_j) + b_i$$

Используя то, что $a_i + a_j + b_i + b_j = \text{const}$, получаем более короткий вариант: $i < j \Leftrightarrow \min(a_j, b_i) > \min(a_i, b_j)$.

2. **Условие.** Даны несколько строк, состоящих из круглых скобок. Нужно сконкатенировать их в некотором порядке так, чтобы конкатенация была правильной скобочной последовательностью.

Решение. *Балансом* назовем количество уже открытых скобок минус количество уже закрытых скобок. Каждая строка характеризуется двумя параметрами — минимальный баланс внутри (или на концах) строки (a_i) и балансом в конце строки (b_i). $a_i \leq 0$, $a_i \leq b_i$. Сперва возьмем в любом порядке все строки, у которых $a_i \geq 0$. Затем будем увеличивать баланс (т.е. возьмем все $b_i \geq 0$). Их нужно отсортировать по убыванию a_i . Остались только строки вида $a_i \leq b_i < 0$. Отсортируем их универсальным компаратором, максимизируя «минимальный баланс». $i < j \Leftrightarrow \min(b_i, a_i + b_j) > \min(b_j, a_j + b_i)$. Если учесть, что $a_i + b_j < b_j$ и $a_j + b_i < b_i$, получается $i < j \Leftrightarrow a_i + b_j > a_j + b_i \Leftrightarrow a_i - b_i > a_j - b_j$.

Сортируем, затем динамика

Давайте усложним «задачу про коробки», «задачу про скобки» и «задачу про заказы с deadline-ами».

1. **«Задача про коробки».** Теперь не обязательно использовать все коробки. Нужно построить башню, используя максимально возможное количество коробок.

Решение. Рассмотрим ответ. Те коробки, которые в него входят, можно брать в порядке, заданном уже разобранным жадностью. Давайте все исходные коробки отсортируем в таком порядке. Тогда башня — какая-то подпоследовательность массива коробок. Будем строить башню сверху вниз. Посчитаем динамику M_{ij} , где i — количество уже рассмотренных коробок, j — сколько из них мы взяли в башню, M_{ij} — минимальная возможная суммарная масса коробок в башне или $+\infty$, если такую башню построить невозможно. Очередную коробку мы можем не включить в башню или, если $w_i \geq M_{ij}$, включить. Ответ на задачу — такое максимальное j : $M_{nj} \neq +\infty$. Получили решение за $O(\text{sort}) + O(n^2)$.

2. **«Задача про скобки».** Теперь не обязательно использовать все строки. Суммарная длина строк не более s . Нужно взять конкатенацию некоторых строк так, чтобы получился префикс некоторой правильной последовательности (т.е. в середине баланс никогда не был меньше нуля, а в конце он может быть положительным), а длина конкатенации была максимально возможной. Решаем также, как и «коробки». B_{ij}

— максимально возможный баланс в конце строки, если мы уже рассмотрели i строк и взяли из них несколько суммарной длины j , или -1 , если нельзя так сделать, не опуская баланс ниже нуля. Ответ на задачу — такое максимальное $j : B_{nj} \neq -1$. Получили решение за $O(\text{sort}) + O(ns)$. Задача доступна на тимусе под номером 1745.

3. **«Заказы с deadline-ами».** В прошлой задаче у каждого заказа был свой $\text{deadline } d_i$ и время выполнения t_i , нужно было успеть выполнить **все** заказы. Решением была сортировка по возрастанию d_i . Новая версия: нужно успеть выполнить как можно больше заказов. Решение = (сортировка по возрастанию d_i) + (динамика T_{ij}), где i — сколько заказов мы уже рассмотрели, j — сколько из них мы уже выполнили, T_{ij} — минимальное суммарное потраченное время.

Избавляемся от динамики

На примере задачи «Заказы с deadline-ами» покажем, как можно решить задачу без динамики и получить решение за $O(n \log n)$.

Способ 1.

Наблюдение: если мы уже выбрали множество заказов, которые собираемся выполнить, можно выполнить их в порядке возрастания d_i . Решение: отсортируем по возрастанию d_i и в таком порядке будем пробовать заказы брать. Если очередной заказ взять не получается (т.е. мы не успеем его выполнить к моменту его deadline), пробуем заменить какой-то уже взятый на наш. Заменять нужно, если $t_i < t_{\max}$, где t_{\max} — максимальное время выполнения по всем уже взятым заказам. Чтобы поддерживать величину t_{\max} , используем `set<int>`.

Способ 2.

Наблюдение: если $\forall i : d_i \geq t_i$ (а если нет, то можно часть заказов сразу удалить), то всегда выгодно выполнить заказ с минимальным t_i (не факт, что первым, но точно выполнить). Доказательство: пусть мы не взяли заказ с минимальным t_i , возьмем первый заказ в порядке выполнения и заменим на заказ с минимальным t_i . Ответ не ухудшился, доказали. Получили решение: сортируем заказы в порядке возрастания t_i . В таком порядке пытаемся добавлять заказы в множество $A = \{\text{заказы, которые точно нужно выполнить}\}$. Чтобы проверить, что можно добавить заказ j в множество A , отсортируем A по d_i и посмотрим, что нет противоречий.

Чтобы проверить за $O(\log n)$, можно ли добавить заказ j , нужно хранить A в декартовом дереве по d_i с функцией в вершине $f_i = d_i - \sum_{j \in \text{левое поддерево}} t_j$, а также функцией в вершине

$$m_i = \min_{j \in \text{поддерево}} f_j.$$

Часть 5. Количество деревьев с точностью до изоморфизма

Помеченные деревья

Пусть все вершины имеют номера $1, 2, \dots, n$, и поэтому различны. Тогда деревья $(1, 2)$, $(2, 3)$ и $(1, 3)$, $(3, 2)$ различаются. Теорема Келли говорит, что количество помеченных деревьев из n вершин равно n^{n-2} . Для доказательства этого факта, можно, например, закодировать дерево Кодом Прюфера: последовательно выбираем лист с наименьшим номером, удаляем его из дерева и записываем номер его отца. Получили массив из $n - 1$ числа, последнее число всегда равно n . Если мы научимся по коду Прюфера однозначно восстанавливать дерево, мы получим, что количество деревьев равно количеству различных кодов Прюфера равно n^{n-2} . Восстановление дерева по коду Прюфера предлагается придумать самостоятельно.

Непомеченные деревья

Классом эквивалентности деревьев назовем максимальное по включению множество попарно изоморфных деревьев. Количество непомеченных деревьев из n -вершин — количество различных классов эквивалентности деревьев из n вершин. Давайте научимся считать это количество. Обозначим через $f_{n,d}$ количество непомеченных корневых деревьев из n вершин глубины d (глубина дерева из одной вершины равна единице).

1. Ответ на задачу выражается через $f_{n,d}$ следующим образом:

$$\sum_{d=1}^{2d \leq n+1} \left((x + \sum_{a=1}^{2a < n} f_{a,d} \cdot f_{n-a,d}) + (f_{n,d} - \sum_{a=1}^{a < n} f_{a,d-1} \cdot F_{n-a,d}) \right)$$

здесь $F_{n,d} = \sum_{i=1}^{i \leq d} f_{n,i}$, а $x = D(f_{n/2,d}, 2)$, если n кратно двум, и нулю в ином случае. $D(n, k)$ — количество способов выбрать из n объектов k упорядоченных объектов с повторениями. $D(n, k) = \binom{n+k-1}{k}$. В этой длинной формуле первые два слагаемых соответствуют случаю «центр дерева — ребро», а вторые два случаю «центр дерева — вершина».

2. У каждого корневого дерева есть две основных характеристики — количество вершин n и глубина d . Назовем пару (d, n) типом дерева. Типы естественным образом упорядочиваются — сперва по d , затем по n . Чтобы посчитать $f_{n,d}$, заведем внутреннюю динамику $g_{n,d,x}$, которая будет набирать в поддереве детей в порядке возрастания типа. $g_{n,d,x}$ — количество способов набрать дерево из n вершин, у корня которого все дети имеют тип строго меньше (d, x) . Тогда $f_{n,d} = g_{n,d,1} - g_{n,d-1,1}$ (все деревья из n вершин высоты строго меньше $d + 1$ минус все деревья

из n вершин высоты строго меньше d , плюс единица берется от того, что мы учли корень дерева).

3. Осталось выписать формулу пересчета для $g_{n,d,x}$.

$$g_{n,d',x'} = \sum_{a=0}^{ax < n} g_{n-xa,d,x} \cdot D(f_{x,d}, a)$$

Здесь (d', x') — следующий тип после (d, x)

$(d' := d, x' := x+1; \text{ if } (x' = \text{maxN}) \{d'++, x' := 0\})$

4. База динамики: $f_{1,1} = 1, g_{1,1,1} = 1$.

Непомеченные деревья, оптимизируем решение

1. С точки зрения олимпиад прежде всего заметим, что все значения можно предподсчитать.
2. Заметим, что динамику g можно разбить по слоям по d (т.к. есть только переходы $d \rightarrow d, d+1$). Сделаем это, теперь у нас квадратная память \Rightarrow кэшируется лучше \Rightarrow работает быстрее.
3. При $a > 0$ и $f_{x,d} = 0$ мы получаем, что $D(f_{x,d}, a) = 0$, поэтому, если $f_{x,d} = 0$, и a уже больше нуля, делаем **break** из цикла по a .
4. D -шки, как и C -шки (сочетания из n по k) быстро пересчитываются. $D(f, k+1) = \binom{f+k}{k+1} = \binom{f+k-1}{k} \cdot \frac{f+k}{k+1} = D(f, k) \cdot \frac{f+k}{k+1}$ Таким образом каждую следующую $D(f_{x,d}, a)$ мы будем получать за $O(1)$.

Теперь можно сказать, что мы используем $O(n^2)$ памяти и $O(n^3 \log n)$ времени, и остановиться (логарифм вылез из условия $ax < n$, количество пар a и x , удовлетворяющих этому ограничению, $O(n \log n)$).

Непомеченные деревья, учимся брать по модулю

В задаче на контесте предлагалось посчитать количество непомеченных деревьев из n вершин по модулю 2^{32} . Складывать, вычитать и умножать по этому модулю очень удобно — считаем все в типе **unsigned int**, не обращая внимания на переполнения. Делить на нечетное число можно используя стандартный метод: $a^{-1} \bmod 2^{32} = a^{\varphi(2^{32})-1} \bmod 2^{32} = a^{2^{31}-1} \bmod 2^{32} = (a^{2^{16}-1})^2 \cdot a \bmod 2^{32} = \dots$ Таким образом мы можем посчитать обратное число за 10 умножений. Вопрос в том, как делить на четное число? Делить на четное число нам может понадобиться только во время

вычисления $\binom{n}{k}$. Причем $\binom{n}{k}$ — целое число, поэтому достаточно научиться только умножать и делить (сложения и вычитания не нужны). Числа вида $2^k a$, где a нечетное, образуют группу по умножению. Умножение: $(k_1, a_1) \cdot (k_2, a_2) = (k_1 + k_2, a_1 \cdot a_2)$, Деление: $(k_1, a_1) / (k_2, a_2) = (k_1 - k_2, a_1 / a_2)$. Поэтому будем считать $\binom{n}{k}$, используя числа именно такого вида. Осталась только одна проблема, по ходу алгоритма нужно считать $\binom{f_{x,d}}{k}$, где $f_{x,d}$ уже взято по модулю 2^{32} . Вопрос: изменится ли остаток «сочетания из n по k » по модулю 2^{32} , если n посчитать по модулю 2^{32} ? Ответ: изменится. Решение проблемы: будем считать все по модулю 2^{64} , утверждается, что остаток по модулю 2^{32} будет посчитан верно :-).

Часть 6. Два указателя в динамике

Задача про профессора и транзисторы

1. **Условие.** У профессора есть k транзисторов и n -этажный дом. Профессор точно знает, что если бросить транзистор с первого этажа, он не разобьется, если его бросить с n -го этажа, то разобьется, а также что функция $f(i)$ «разобьется ли транзистор, если его бросить с i -го этажа» монотонна. Помогите профессору за минимальное количество бросков транзисторов в худшем случае определить такое i , что $x(i) = 0, x(i + 1) = 1$.
2. **Решение.** Пусть $f_{n,k}$ — ответ на задачу с параметрами (n, k) . По условию $f_{2,k} = 0, f_{n,k} = \min_{i=2..n-1} \max(f_{i,k-1}, f_{n-i+1,k})$. Если посчитать в лоб, получается $O(n^2 k)$.
3. **Мысль 1.** Если $k \geq \log_2 n$, то можно делать двоичный поиск, поэтому можно сделать в начале $k = \min(k, \lceil \log_2 n \rceil)$. Теперь решение работает за $O(n^2 \log n)$.
4. **Мысль 2.** $f_{i,k} \leq f_{i+1,k}$, поэтому $f_{i,k-1} \downarrow, f_{n-i+1,k} \uparrow$, и минимум функции $\max(f_{i,k-1}, f_{n-i+1,k})$ можно найти бинарным поиском (ищем корень функции $f_{i,k-1} - f_{n-i+1,k}$). Теперь решение работает за $O(n \log^2 n)$.
5. **Мысль 3.** Обозначим за $p_{n,k}$ оптимальное i , которое используется в формуле пересчета. Тогда $p_{n,k} \leq p_{n+1,k}$ и можно использовать метод двух указателей: изначально делаем $p_{n+1,k} = p_{n,k}$ и, пока функция не ухудшается, увеличиваем $p_{n+1,k}$ на 1. Теперь решение работает за $O(n \log n)$.

6. **Эпилог.** На самом деле данная задача имеет решение для $n, k \leq 10^{18}$, для этого достаточно заметить, что $f_{n,1} = n-2$, а $f_{n,2} \approx n^{\frac{1}{2}}$, а $f_{n,k} \approx n^{\frac{1}{k}}$, более-менее точно подобрать формулы (ограничения снизу и сверху) и считать динамику лениво, пропуская все заведомо неоптимальные состояния.

Разбор задачи про суммы (Day1-G)

1. **Условие.** Есть упорядоченный массив из n чисел, нужно выбрать k из них так, чтобы сумма расстояний от каждого из n чисел до ближайшего из k выбранных была минимально возможной.
2. **Решение.** Заметим, что массив из n чисел в результате разобьется на k отрезков. Ответ для одного отрезка можно посчитать за $O(1)$ — нужно выбрать среднее число (если чисел на отрезке k , берем число с номером $\frac{k}{2}$, округляем в любую сторону), когда число выбрано, сумму мы получаем за $O(1)$, используя предподсчитанные частичные суммы. Пусть $f_{i,j}$ — минимальная сумма расстояний, которая может получиться, если первые i чисел разбить на j отрезков. База этой динамики уже есть, мы знаем ответы для $j = 1$. Сделаем переход:

$$f_{i,j} = \min_{p=1\dots i} (f_{p,j-1} + s_{p+1,i})$$

где $s_{l,r}$ — значение суммы для отрезка $[l\dots r]$. Если отрезок пуст, $s_{l,r} = 0$. Данная динамика имеет $O(nk)$ состояний и считается за время $O(n^2k)$.

3. **Оптимизируем решение.** $p_{n+1,k} \geq p_{n,k}$. Иначе, откинув последнее число, мы бы получили более хорошее разбиение на отрезки для состояния (n, k) . Аналогично $p_{n,k} \geq p_{n-1,k}$. Теперь менее тривиальный факт: $p_{n,k-1} \leq p_{n,k} \leq p_{n,k+1}$. Короткое доказательство я к сожалению привести не могу. Тем не менее, насчитав все $p_{n,k}$ и $f_{n,k}$, этот факт легко проверить на существующих тестах. Из двух вышеприведенных неравенств получаем следующее:

$$p_{n,k-1} \leq p_{n,k} \leq p_{n+1,k}$$

Давайте будем перебирать $p_{n,k}$ ровно в таком диапазоне. Можно считать $p_{n+1,i} = n$. Суммарное время работы =

$$\sum_{i=2}^k \sum_{j=1}^n (p_{j+1,i} - p_{j,i-1} + 1) = O(n^2)$$

Заметим, что почти все $p_{i,j}$ присутствуют в сумме, как со знаком

плюс, так и со знаком минус. Т.е., все успешно сократится. Поэтому $O(n^2)$. Покажем также, что при $k \geq 2$ время работы может быть порядка n^2 . Пусть $k = 2$, $a_i = i$, тогда выгодно делить равномерно, т.е. $p_{n,k} = n - \frac{n}{k}$. Все значения при $k = 1$ посчитаны за $O(1)$, посчитаем время потраченное для $k = 2$: $\sum_{j=1}^n (p_{j+1,2} - p_{j,1} + 1) = \sum_{j=1}^n (\frac{j+1}{2} - 0 + 1) \approx \frac{n^2}{4}$.

Разбор задачи *command-post*

1. **Условие.** Даны n различных точек на окружности, нужно выбрать $3 \leq k$ из них так, чтобы площадь получившегося многоугольника была максимально возможной, и центр окружности обязательно лежал внутри. Второе условие в принципе можно опустить, так как, если центр не лежит внутри, то, или невозможно поместить центр внутрь многоугольника, или текущий многоугольник не максимален.

2. **Решение.** Состояние: мы взяли точку l , точку r , и на отрезке $(l \dots r)$ хотим взять еще k точек. Начальное состояние $[i, i, k - 1]$ для всех i . Пара (i, i) означает, что мы должны сделать полный круг. Переход:

$$f_{l,r,k} = \max_{m=l+1..r-1} (f_{l,m,0} + f_{m,r,k-1})$$

Получили решение за $O(n^3 k)$.

3. **Мысль 1.** Давайте разбивать k точек не на 1 и $k - 1$, а на $k/2$ и $k - k/2$. Тогда различных k будет $O(\log k)$, а суммарное время работы $O(n^3 \log k)$.

4. **Мысль 2.** Оптимальное m будем запоминать в $p_{l,r}$. Утверждается, что $p_{l,r-1} \leq p_{l,r} \leq p_{l+1,r}$. Будем перебирать $p_{l,r}$ в указанном промежутке. Получим суммарное время работы для каждого k равное $\sum_{l,r} (p_{l+1,r} - p_{l,r-1})$. Почти все $p_{i,j}$ присутствуют в сумме, как со знаком плюс, так и со знаком минус, поэтому сумма равна $O(n^2)$. Получили асимптотику $O(n^2 \log k)$.

Часть 7. Разбиение числа на слагаемые

Условие задачи: для фиксированного числа n посчитать количество разбиений этого числа на слагаемые. При этом разбиения $1 + 2$ и $2 + 1$ считаются одинаковыми.

Максимальное слагаемое не более k

Если можно использовать одинаковые слагаемые, то

$$f_{n,k} = f_{n-k,k} + f_{n,k-1}$$

и решение выглядит так:

```
1 f[0] = 1
2 for (x = 1; x <= k; x++)
3   for (i = 0; i + x <= n; i++)
4     f[i + x] += f[i]
```

Если нельзя использовать одинаковые слагаемые, то

$$f_{n,k} = f_{n-k,k} + f_{n-k,k-1}$$

и решение выглядит так:

```
1 f[0] = 1
2 for (x = 1; x <= k; x++)
3   for (i = n - x; i >= 0; i--)
4     f[i + x] += f[i]
```

Количество слагаемых не более k

Если можно использовать одинаковые слагаемые, то

$$f_{n,k} = f_{n-k,k} + f_{n,k-1}$$

Доказательство: или уменьшили все слагаемые на 1, или сказали, что меньшее из слагаемых равно нулю. Если нельзя использовать одинаковые слагаемые, то

$$f_{n,k} = f_{n-k,k} + f_{n-k,k-1}$$

Почему получились формулы такие же, как для двух предыдущих задач? Если слагаемые нарисовать подряд идущими клетчатыми столбцами, причем высота очередного столбца равна величине соответствующего слагаемого, то получится так называемая диаграмма Юнга. Заметим, что ее можно разворачивать на 90 градусов (смотреть на нее под другим углом). Количество разбиений числа n на слагаемые равно количеству диаграмм Юнга из n клеток. Количество столбцов в диаграмме равно количеству слагаемых, высота максимального столбца в диаграмме равно максимальному слагаемому. При повороте на 90 градусов понятия «количество столбцов» и «высота максимального столбца» поменяются местами.

Не более k и ровно k

Если f_k — ровно k , а F_k — не более k , то $f_k = F_k - F_{k-1}$, а $F_k = \sum_i f_i$

к различных слагаемых

Задачу про k различных слагаемых можно свести к задаче про произвольные слагаемые. Для этого нужно из i -го слагаемого вычесть i . Получилось, что теперь число $n - \frac{k(k+1)}{2}$ нужно разбить на k произвольных слагаемых, а заодно мы получили, что $k \leq \sqrt{n}$, поэтому задача решается за $O(n\sqrt{n})$.

Часть 8. Эквивалентность некоторых задач про жадность

Сформулируем задачи и введем обозначения:

1. **«Коробки»**. У каждой коробки есть масса m_i и максимальный вес w_i , который коробка может выдержать. Нужно построить максимальную по количеству коробок башню.
2. **«Заказы с deadline-ами»**. У каждого заказа есть deadline d_i и время выполнения t_i . Нужно максимальное количество заказов успеть выполнить до наступления deadline-а.
3. **«Школьники в яме»**. Школьники пытаются выбраться из ямы глубины H . У каждого школьника есть два параметра — длина рук l_i и высота h_i . Школьники могут становиться друг на друга (выносливость школьника бесконечна). Если все школьники выстроились в башню и $l_i + \sum h_j \geq H$, то верхний школьник может выбраться из ямы. Нужно придумать такой порядок, в котором школьники выбираются из ямы, что спастись сможет максимальное количество школьников.
4. **«Авторитеты»** У Толика есть изначальный авторитет A . Есть n людей. i -й человек навсегда присоединится к Толику, если авторитет Толика хотя бы a_i и Толик его пригласит. После того, как i -й человек присоединится к Толику, авторитет Толика увеличится на b_i . И a_i , и b_i — целые, возможно отрицательные, числа. Нужно выбрать стратегию для Толика, чтобы в итоге количество его сторонников было максимально возможным.

«Коробки» \leftrightarrow «Заказы с deadline-ами»

$$\begin{aligned} \sum m_j &\leq w_i \\ m_i + \sum m_j &\leq w_i + m_i \\ d_i &= w_i + m_i, t_i = m_i \text{ (все } t_i > 0, \text{ все } m_i > 0) \end{aligned}$$

«Коробки» \leftrightarrow «Школьники в яме»

Обозначим $\sum h_i$ за S , рассмотрим людей сверху вниз.

$l_i + (S - \sum h_j) \geq H$ (i — текущий человек, j — люди над ним)

$(S - H) + l_i \geq \sum h_j$ (перенесли и сгруппировали)

Получили $m_i = h_i$, $w_i = l_i + (S - H)$ (все $m_i > 0$, все $h_i > 0$)

«Коробки» \leftrightarrow «Авторитеты»

Задача про авторитеты решается в два этапа. Сперва пытаемся взять всех людей, повышающих авторитет ($b_i \geq 0$). Их мы просто отсортируем по возрастанию a_i и в таком порядке будем пытаться брать. Теперь у нас есть некоторый авторитет A , и каждый очередной человек уменьшает авторитет ($b_i < 0$).

$A + \sum b_j \geq a_i$ (i — текущий человек, j — люди с отрицательным b_j , которых мы уже взяли)

$-\sum b_j \leq A - a_i$

Получили $w_i = A - a_i$, $m_i = -b_i$ (все $m_i > 0$, все $b_i < 0$)

Эпилог

Поскольку в конце главы про жадность мы уже показали, что задача «Заказы с deadline-ами» решается за $O(n \log n)$ с использованием только сортировки и `set<int>`, можно сделать вывод, что задачи «Коробки», «Школьники в яме», «Авторитеты» решаются такой же жадностью за время $O(n \log n)$.

Задачи и разборы

Задача А. Белоснежка и n гномов

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

«Ну не гномы, а наказание какое-то!», — подумала Белоснежка, в очередной раз пытаясь уложить гномов спать. Одного уложишь — другой уже проснулся! И так всю ночь.

У Белоснежки n гномов, и все они очень разные. Она знает, что для того, чтобы уложить спать i -го гнома нужно a_i минут, и после этого он будет спать ровно b_i минут. Помогите Белоснежке узнать, может ли она

получить хотя бы минутку отдыха, когда все гномы будут спать, и если да, то в каком порядке для этого нужно укладывать гномов спать.

Например, пусть есть всего два гнома, $a_1 = 1$, $b_1 = 10$, $a_2 = 10$, $b_2 = 20$. Если Белоснежка сначала начнет укладывать первого гнома, то потом ей потребуется целых 10 минут, чтобы уложить второго, а за это время проснется первый. Если же она начнет со второго гнома, то затем она успеет уложить первого и получит целых 10 минут отдыха.

Формат входного файла

Первая строка входного файла содержит число n ($1 \leq n \leq 10^5$), вторая строка содержит числа a_1, a_2, \dots, a_n , третья — числа b_1, b_2, \dots, b_n ($1 \leq a_i, b_i \leq 10^9$).

Формат выходного файла

Выведите в выходной файл n чисел — порядок, в котором нужно укладывать гномов спать. Если Белоснежке отдохнуть не удастся, выведите число -1 .

Пример

stdin	stdout
2 1 10 10 20	2 1
2 10 10 10 10	-1

Разбор задачи А. Белоснежка и n гномов

Сортировка по $a_i + b_i$.

Задача В. Эльфы и олени

Вход: stdin
 Выход: stdout
 Ограничение по времени: 0.5 с
 Ограничение по памяти: 256 Мб

Скоро новый год и Санта-Клаус уже начал готовить свою волшебную оленью упряжку, на которой он развозит подарки детям. Известно, что

упряжку везут несколько волшебных оленей, на каждом из которых едут два эльфа.

Но волшебные олени — строптивые животные, поэтому не любые два эльфа могут ехать на любом олене. А именно, каждый олень характеризуется некоторой строптивостью a_i , а каждый эльф — темпераментом b_i . Два эльфа j и k могут ехать на i -м олене в том и только в том случае, если либо $b_j < a_i < b_k$, либо $b_k < a_i < b_j$.

Чтобы его появление было максимально зрелищным, Санта-Клаус хочет, чтобы в его упряжке было как можно больше оленей. Про каждого оленя Санта знает его строптивость, а про каждого эльфа — его темперамент.

Помогите Санте выяснить, какое максимальное количество оленей он сможет включить в упряжку, каких оленей ему следует выбрать, и какие эльфы должны на них ехать.

Формат входного файла

Первая строка входного файла содержит два целых числа m и n — количество оленей и эльфов, соответственно ($1 \leq m, n \leq 100\,000$).

Вторая строка содержит m целых чисел a_i — строптивость оленей ($0 \leq a_i \leq 10^9$). Третья строка содержит n целых чисел b_i — темперамент эльфов ($0 \leq b_i \leq 10^9$).

Формат выходного файла

На первой строке выходного файла выведите одно число k — максимальное количество оленей, которое Санта-Клаус может включить в свою упряжку. На следующих k строках выведите по три целых числа: d_i , $e_{i,1}$, $e_{i,2}$ — для каждого оленя в упряжке выведите его номер и номера эльфов, которые на нем поедут. Если решений несколько, выведите любое. Выводить эльфов нужно в таком порядке, что $b_{e_{i,1}} < a_{d_i} < b_{e_{i,2}}$.

И эльфы, и олени пронумерованы, начиная с единицы, в том порядке, в котором они заданы во входном файле.

Пример

stdin	stdout
4 6	2
2 3 4 5	1 1 2
1 3 2 2 5 2	3 4 5

Разбор задачи В. Эльфы и Олени

Отсортируем массивы a и b по возрастанию. Запускаем бинарный поиск по ответу. Чтобы проверить, что ответ больше либо равен k , говорим, что

i -я пара эльфов это (b_i, b_{n-k+i}) , и запускаем метод двух указателей по «массиву оленей» и «массиву пар эльфов».

Задача С. Приключение

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Теплым весенним днем группа из N школьников-программистов гуляла в окрестностях города Кисловодска. К несчастью, они набрали на большую и довольно глубокую яму. Как это случилось – непонятно, но вся компания оказалась в этой яме.

Глубина ямы равна H . Каждый школьник знает свой рост по плечи h_i и длину своих рук l_i . Таким образом, если он, стоя на дне ямы, поднимет руки, то его ладони окажутся на высоте $h_i + l_i$ от уровня дна ямы. Школьники могут, вставая друг другу на плечи, образовывать вертикальную колонну. При этом любой школьник может встать на плечи любого другого школьника. Если под школьником i стоят школьники j_1, j_2, \dots, j_k , то он может дотянуться до уровня $h_{j_1} + h_{j_2} + \dots + h_{j_k} + h_i + l_i$.

Если школьник может дотянуться до края ямы (то есть $h_{j_1} + h_{j_2} + \dots + h_{j_k} + h_i + l_i \geq H$), то он может выбраться из нее. Выбравшиеся из ямы школьники не могут помочь оставшимся. Найдите наибольшее количество школьников, которые смогут выбраться из ямы до прибытия помощи, и перечислите их номера.

Формат входного файла

В первой строке входного файла записано натуральное число N – количество школьников, попавших в яму. Далее в N строках указаны по два целых числа: рост i -го школьника по плечи h_i и длина его рук l_i . В последней строке указано целое число – глубина ямы H .

Формат выходного файла

В первой строке выведите K – максимальное количество школьников, которые смогут выбраться из ямы. Если $K > 0$, то во второй строке в произвольном порядке выведите их номера, разделяя их пробелами. Школьники нумеруются с единицы в том порядке, в котором они заданы во входном файле. Если существует несколько решений, выведите любое.

Система оценки

Подзадача 1 (50 баллов) $n \leq 2\,000$ $1 \leq l_i, h_i, H \leq 10^5$

Подзадача 2 (50 баллов) $n \leq 100\,000$ $1 \leq l_i, h_i, H \leq 10^9$

Пример

stdin	stdout
2 10 4 5 2 20	0
6 6 7 3 1 8 5 8 5 4 2 10 5 30	4 1 4 2 5

Разбор задачи С. Приключение

См. лекцию. «Часть 8, задача 3», а также «Часть 4, задача про заказы с deadline-ами».

Задача D. Авторитеты

Вход: stdin
 Выход: stdout
 Ограничение по времени: 0.5 с
 Ограничение по памяти: 256 Мб

Толик придумал новую технологию программирования. Он хочет уговорить друзей использовать ее. Однако все не так просто. i -й друг согласится использовать технологию Толика, если его авторитет будет не меньше a_i (авторитет выражается целым числом). Как только i -й друг начнет ее использовать, к авторитету Толика прибавится число b_i (попадаются люди, у которых $b_i < 0$). Помогите Толику наставить на путь истинный как можно больше своих друзей.

Формат входного файла

На первой строке содержатся два целых числа. Количество друзей у Толика n и первоначальный авторитет Толика a_0 ($-10^9 \leq a_0 \leq 10^9$). Следующие n строк содержат пары целых чисел a_i и b_i ($-10^9 \leq a_i, b_i \leq 10^9$).

Формат выходного файла

На первой строке выведите число m — максимальное число друзей, которых может увлечь Толик. На второй строке выведите m чисел — номера друзей в том порядке, в котором их нужно агитировать.

Система оценки

Подзадача 1 (50 баллов) $1 \leq n \leq 10^3$

Подзадача 2 (50 баллов) $1 \leq n \leq 10^5$

Пример

stdin	stdout
5 1	4
1 3	1 4 3 5
6 -5	
6 -4	
2 2	
2 -1	

Разбор задачи D. Авторитеты

См. лекцию. «Часть 8, задача 4».

Задача E. Коробки

Вход: stdin
 Выход: stdout
 Ограничение по времени: 0.5 с
 Ограничение по памяти: 256 Мб

У Васи в комнате очень много коробок, которые валяются в разных местах. Васина мама хочет, чтобы он прибрался. Свободного места в комнате мало и поэтому Вася решил собрать все коробки и составить их одну на другую.

К сожалению, это может быть невозможно. Например, если на картонную коробку с елочными украшениями положить что-то железное и

тяжелое, то вероятно следующий Новый год придется встречать с новыми игрушками.

Вася взвесил каждую коробку и оценил максимальный вес который она может выдержать. Помогите ему определить какое наибольшее количество коробок m он сможет поставить одну на другую так, чтобы для каждой коробки было верно, что суммарный вес коробок сверху не превышает максимальный вес, который она может выдержать.

Формат входного файла

Первая строка входного файла содержит целое число n ($1 \leq n$) — количество коробок в комнате. Каждая следующая из n строк содержит два целых числа w_i и c_i ($1 \leq w_i \leq 10^5, 1 \leq c_i \leq 10^9$), где w_i — это вес коробки с номером i , а c_i — это вес который она может выдержать.

Формат выходного файла

В выходной файл выведите одно число — ответ на задачу.

Система оценки

Подзадача 1 (50 баллов) $n \leq 1250$

Подзадача 2 (50 баллов) $n \leq 100\,000$

Пример

stdin	stdout
3 10 11 20 100 30 10	3
3 11 11 20 100 30 10	2

Разбор задачи Е. Коробки

См. лекцию. «Часть 8, задача 1».

Задача F. Простые пути в дереве

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Дан неориентированный связный граф из n вершин и $n - 1$ ребра. Требуется для каждого ребра посчитать суммарную длину простых путей, проходящих через это ребро. Длиной пути здесь называется количество ребер в пути.

Формат входного файла

На первой строке целое число n ($2 \leq n \leq 300\,000$). Следующие $n - 1$ строка содержат пары чисел от 1 до n — ребра графа.

Формат выходного файла

$n - 1$ строка. i -я строка должна содержать целое число — ответ для i -го ребра.

Система оценки

Подзадача 1 (50 баллов) $n \leq 3\,000$.

Подзадача 2 (50 баллов) $n \leq 300\,000$.

Пример

stdin	stdout
5	13
1 2	8
2 3	8
2 4	9
5 1	

Разбор задачи F. Простые пути в дереве

Подвесим дерево за первую вершину. Первым `dfs`-ом насчитаем для каждой вершины v $s_1[v]$ — количество вершин в поддереве вершины v , и $s_2[v]$ — суммарную длину всех путей от вершины v вниз. Вторым `dfs`-ом по дереву для каждого ребра насчитаем ответ. Каждое ребро имеет два конца — нижний конец a и верхний конец b . Количество вершин снизу — $s_1[a]$, количество вершин сверху $t_1[b] = n - s_1[a]$, суммарная длина путей вниз — $s_2[a]$, суммарную длину путей вверх $t_2[b]$ можно пересчитывать,

при спуске вниз: $t_2[a] = t_2[b] + t_1[b] + s_2[b] - (s_2[a] + s_1[a])$. База: $t_2[root] = 0$.
 Ответ для ребра (a, b) выражается так: $s_1[a] \cdot t_2[b] + s_2[a] \cdot t_1[b] + s_1[a] \cdot t_1[b]$.

Задача Г. Редукция дерева

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Задано неориентированное дерево, содержащее n вершин. Можно выбрать некоторое ребро и удалить его, при этом инцидентные ему вершины не удаляются. Таким образом можно удалить из дерева некоторый набор рёбер. В результате дерево распадается на некоторое количество меньших деревьев. Требуется, удалив наименьшее количество рёбер, получить в качестве хотя бы одной из компонент связности дерево, содержащее ровно p вершин.

Формат входного файла

Первая строка входного файла содержит пару натуральных чисел n и p ($1 \leq p \leq n \leq 1000$). Далее в $n - 1$ строке содержатся описания рёбер дерева. Каждое описание состоит из пары натуральных чисел a_i, b_i ($1 \leq a_i, b_i \leq n$) — номеров соединяемых ребром вершин.

Формат выходного файла

В первую строку выведите наименьшее количество рёбер q в искомом наборе. Во вторую строку выведите номера удаляемых рёбер. Номера рёбер определяются порядком их задания во входном файле. Рёбра нумеруются с единицы. Если оптимальных решений несколько, разрешается выводить любое.

Система оценки

Подзадача 1 (50 балла) $1 \leq p \leq n \leq 200$.
Подзадача 2 (50 балла) $1 \leq p \leq n \leq 1000$.

Пример

stdin	stdout
11 6	2
1 2	3 6
1 3	
1 4	
2 6	
2 7	
1 5	
2 8	
4 9	
4 10	
4 11	

Разбор задачи G. Редукция дерева

Подвесим дерево. Для каждой вершины v насчитаем $f[v, k]$ — сколько ребер в поддереве вершины v нужно разрезать, чтобы компонента связности с корнем в вершине v имела размер ровно k . Для удобства пересчета и оценки времени работы предположим, что дерево бинарное. В противном случае нужно будет делать внутреннюю динамику по детям вершины v . Итак, дерево бинарное, переход делается так:

```
for (l = 1; l <= left_size; l++)
    for (r = 1; r <= right_size; r++)
        f[v, l+r] = min(f[v, l+r], f[left, l] + f[right, r])
```

Время работы алгоритма $\sum_{v=1..n} (\text{left_size}[v] \cdot \text{right_size}[v])$. Утверждается, что эта сумма $O(n^2)$. Подробнее смотрите по адресу <http://codeforces.ru/blog/entry/6703#comment-122804>

Задача H. Изоморфные деревья

Вход: stdin
 Выход: stdout
 Ограничение по времени: 0.2 с
 Ограничение по памяти: 256 Мб

Два дерева из n вершин называются *изоморфными*, если существует такая перестановка вершин второго дерева, что после применения данной

перестановки ко второму дереву, его матрица смежности совпадет с матрицей смежности первого дерева. Несложно проверить, что данное отношение транзитивно.

Классом эквивалентности деревьев назовем максимальное по включению множество попарно изоморфных деревьев. Требуется посчитать количество различных классов эквивалентности деревьев из n вершин. Данное число может быть очень большим, достаточно посчитать лишь его остаток по модулю 2^{32} .

Формат входного файла

На первой строке целое число n .

Формат выходного файла

Количество различных классов эквивалентности деревьев из n вершин по модулю 2^{32} .

Система оценки

- Подзадача 1 (20 баллов)** $n \leq 10$.
Подзадача 2 (20 баллов) $n \leq 18$.
Подзадача 3 (20 баллов) $n \leq 28$.
Подзадача 4 (20 баллов) $n \leq 300$.
Подзадача 5 (20 баллов) $n \leq 2000$.

Пример

stdin	stdout
3	1
5	3
7	11

Примечание



Разбор задачи Н. Изоморфные деревья

См. лекцию. «Часть 5».

Задача I. К минимумов на отрезке

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 1 с
 Ограничение по памяти: 256 Мб

Дан массив a из n целых чисел и q запросов вида «вывести k первых чисел в отсортированной версии отрезка $[l \dots r]$ нашего массива».

Пример: $n = 7$, $a = [6, 1, 5, 2, 4, 3, 1]$, $l = 2$, $r = 4$, $k = 2$. Отрезок $[l \dots r] = [1, 5, 2]$. Его отсортированная версия = $[1, 2, 5]$. Первые 2 числа = $[1, 2]$.

Формат входного файла

На первой строке число n ($1 \leq n \leq 100\,000$).

На второй строке массив a (n целых чисел от 1 до 10^9).

На третьей строке количество запросов q ($1 \leq q \leq 100\,000$).

Следующие q строк содержат тройки чисел $l_i \ r_i \ k_i$

$1 \leq l_i \leq r_i \leq n$, $1 \leq k_i \leq \min(r_i - l_i + 1, 10)$

Формат выходного файла

Для каждого из q запросов выведите ответ (k_i чисел) на отдельной строке. Числа внутри одного запроса нужно выводить в порядке возрастания. Для лучшего понимания условия и формата данных смотрите пример.

Система оценки

Подзадача 1 (50 баллов) $n, q \leq 100\,000$ $l_i \leq l_{i+1}, r_i \leq r_{i+1}$.

Подзадача 2 (50 баллов) $n, q \leq 30\,000$ l_i и r_i произвольны.

Пример

stdin	stdout
7	1 1 2 3 4 5 6
6 1 5 2 4 3 1	1 2
4	2
1 7 7	1 3
2 4 2	
3 5 1	
5 7 2	

Разбор задачи I. К минимумов на отрезке

См. лекцию. «Часть 2».

Операции L++, L--, R++, R-- делаем с помощью `multiset<int>`

Получаем решение за $O((n + q)\sqrt{n} \log n + qk)$

Задача J. Инверсии отрезка

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 3 с
 Ограничение по памяти: 256 Мб

Дана перестановка p из n чисел. Ваша задача — быстро обрабатывать запросы вида *количество инверсий на отрезке данной перестановки*. Напомним, инверсией называется пара индексов (i, j) : $i < j$, $p_i > p_j$.

Формат входного файла

На первой строке число n .

На второй строке сама перестановка.

На третьей строке количество запросов q .

Следующие q строк содержат пары чисел l_i, r_i ($1 \leq l_i \leq r_i \leq n$).

Формат выходного файла

Для каждого из q запросов выведите ответ на отдельной строке.

Система оценки

Подзадача 1 (50 баллов) $1 \leq n, q \leq 10^5$ $l_i \leq l_{i+1}, r_i \leq r_{i+1}$.

Подзадача 2 (50 баллов) $1 \leq n, q \leq 20\,000$ l_i и r_i произвольны

Пример

stdin	stdout
5	3
5 4 3 1 2	9
3	2
1 3	
1 5	
3 5	

Разбор задачи J. Инверсии отрезка

См. лекцию. «Часть 2».

Множество чисел, соответствующее отрезку $[L..R]$ храним в дереве Фенвика. Операции $L++$, $L--$, $R++$, $R--$, обрабатываем с помощью дерева Фенвика за $O(\log n)$.

Задача K. Продуктивный бинпоиск

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	512 Мб

Вам дан следующий вариант бинарного поиска:

```
1 int l = m, r = n, result = n + 1;
2 while (l <= r) {
3     int c = f(l, r);
4     if (enough(c)){
5         result = c;
6         r = c - 1;
7     } else {
8         l = c + 1;
9     }
10 }
```

Процедура `enough(c)` возвращает `true` или `false`. Известно, что она монотонна. Т.е. для всех $c < c_0$ она вернет `false`, а для всех $c \geq c_0$ она вернет `true`. Стоимость вызова процедуры `enough(x)` равна x . Нужно выбрать такую функцию f , чтобы суммарная стоимость вызова одного бинпоиска в худшем случае была минимально возможной.

Формат входного файла

Два целых числа m и n ($1 \leq m \leq n \leq 6\,500$) — границы в бинпоиске.

Формат выходного файла

Одно целое число — минимальную суммарную стоимость вызова бинпоиска в худшем случае при использовании оптимальной функции f .

Система оценки

Подзадача 1 (50 баллов) $n \leq 2\,000$.

Подзадача 2 (50 баллов) $n \leq 6\,500$.

Примеры

stdin	stdout
1 1	1
1 5	9

Примечание

Во втором примере выгодно сперва вызвать `enough(4)`.

Затем, или `enough(5)`, или `enough(2)`.

И затем (для последовательности 4-2-?), или `enough(3)`, или `enough(1)`.

Возможные стоимости: $4+5 = 9$, $4+2+3 = 9$, $4+2+1 = 7$, максимум = 9.

Разбор задачи К. Продуктивный бинпоиск

Сперва выпишем динамику по подотрезкам за $O(n^3)$.

$$f_{l,r} = \min_{m=l+1\dots r-1} (\max(f_{l,m-1}, f_{m+1,r}) + m)$$

Заметим, что $f_{l+1,r} \leq f_{l,r} \leq f_{l,r+1}$. Поэтому $f_{l,m-1}$ возрастает по m , а $f_{m+1,r}$ убывает по m . Найдем минимальное p : $f_{p+1,r} \leq f_{l,p-1}$. Позиции, большие, чем p , рассматривать не имеет смысла, так как $(f_{l,m-1} + m)$ возрастает на $[p \dots r]$. Получаем:

$$f_{l,r} = \min(f_{l,p-1} + p, \min_{m=l+1\dots p-1} (f_{m+1,r} + m))$$

Зафиксируем l . Будем перебирать r в порядке убывания. Позиция $p_{l,r}$ при этом убывает и пересчитывается следующим образом:

```
while (p - 1 > 1 && f[l][p - 2] > f[p][r]) p--;
```

Минимум легко считается за $O(1)$, так как границы отрезка, на котором нужно считать минимум, $l+1$ и $p-1$, обе убывают. Получили решение за $O(n^2)$.

Задача Л. Командный пункт

Вход: stdin
 Выход: stdout
 Ограничение по времени: 1.5 с
 Ограничение по памяти: 512 Мб

Полковник Кругляковски только что закончил свою последнюю кампанию. Сейчас он установил свой командный пост в центре прекрасной долины и хочет защитить себя и свою армию от возможных атак.

Чтобы добиться защищённости, нужно построить несколько наблюдательных пунктов. Кругляковски уже отправил своих лучших скаутов, которые по возвращении доложили ему список из n наиболее подходящих мест для постройки наблюдательных пунктов. Все n найденных мест располагаются на одинаковом расстоянии от командного пункта (то есть на окружности с центром в точке расположения командного поста). К своему сожалению, Кругляковски быстро понял, что ресурсов хватит на постройку только k наблюдательных пунктов. Кругляковски также планирует построить забор, соединив соседние наблюдательные пункты.

Пожалуйста, помогите Колонэлю выбрать k позиций для наблюдательных постов таким образом, чтобы максимизировать площадь, окружённую забором. При этом командный пункт должен оказаться внутри забора, или на границе. Вы можете предположить, что земля плоская, пренебречь размером командного пункта и наблюдательных постов. Забор — это выпуклый многоугольник с вершинами в наблюдательных постах.

Формат входного файла

Первая строка содержит два целых числа n и k ($3 \leq k \leq n \leq 1000$). Следующие n чисел содержат полярные углы (в радианах) возможных наблюдательных постов. Углы даны на отдельных строках. Углы задаются вещественными числами с не более чем четырьмя знаками после запятой. Все углы находятся в диапазоне от 0 до 2π . Все углы различны. Командный пост находится в точке $(0, 0)$.

Формат выходного файла

Выведите k различных целых чисел от 1 до n — индексы точек, которые вы используете для постройки наблюдательных постов. Точки нужно перечислить в порядке увеличения полярного угла. Если есть несколько оптимальных решений, выведите любое. Если же защитить командный пункт с помощью забора невозможно, выведите вместо этого 0 на отдельной строке.

Система оценки

Подзадача 1 (50 баллов) $n \leq 300$

Подзадача 2 (50 баллов) $n \leq 1000$

Примеры

stdin	stdout
4 4 1.57 0 3.14 4.71	2 1 3 4
4 3 1.57 0 3.14 4.71	2 1 4

Разбор задачи L. Командный пункт

См. лекцию. «Часть 6, разбор задачи command-post».

Задача M. Общая подпоследовательность

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 2 с
 Ограничение по памяти: 16 Мб

Даны две строки, состоящих из маленьких латинских букв. Нужно найти их наибольшую общую подпоследовательность.

Формат входного файла

На первой строке первая строка.
 На второй строке вторая строка.
 Длины строк от 1 до 5000.

Формат выходного файла

Максимальную по длине общую подпоследовательность на отдельной строке. Если ответов несколько, выведите любой. Если ответ пуст, перевод строки выводить все равно нужно.

Система оценки

Подзадача 1 (50 баллов) Длины строк до 1000.
Подзадача 2 (50 баллов) Длины строк до 5 000.

Пример

stdin	stdout
abacabadabacaba dbdcccdbd	bccb

Разбор задачи М. Общая подпоследовательность

См. лекцию. «Часть 3».

Задача N. Различные подпоследовательности

Вход: stdin
 Выход: stdout
 Ограничение по времени: 0.5 с
 Ограничение по памяти: 256 Мб

Дана последовательность целых чисел длины n . Нужно найти количество различных непустых подпоследовательностей.

Система оценки

Подзадача 1 (25 баллов) $1 \leq n \leq 10$ $1 \leq a_i \leq 10$
Подзадача 2 (25 баллов) $1 \leq n \leq 10^4$ $1 \leq a_i \leq 10$
Подзадача 3 (25 баллов) $1 \leq n \leq 10^5$ $1 \leq a_i \leq 100$
Подзадача 4 (25 баллов) $1 \leq n \leq 3 \cdot 10^5$ $1 \leq a_i \leq 3 \cdot 10^5$

Формат входного файла

На первой строке n . На второй строке n целых чисел.

Формат выходного файла

Количество различных подпоследовательностей по модулю $10^9 + 7$.

Пример

stdin	stdout
3 1 1 2	5
35 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5	941167856

Разбор задачи N. Различные подпоследовательности

$f[i]$ — количество уникальных последовательностей, конец которых — в точности i -й элемент массива. $prev[i]$ — ближайшая слева позиция, такая что: $a[prev[i]] = a[i]$. Следующий код решает задачу, и может быть реализован за $O(n)$ с использованием частичных сумм.

```
1 f[0] = 1
2 for (i = 1; i <= n; i++)
3   f[i] = сумма f[prev[i]..i-1]
4 result = сумма f[1..n]
```

Задача O. Наибольшая общая возрастающая

Вход: `stdin`
 Выход: `stdout`
 Ограничение по времени: 0.5 с
 Ограничение по памяти: 256 Мб

Даны две последовательности чисел. Нужно найти наибольшую общую возрастающую подпоследовательность. Более формально: такие $1 \leq i_1 < i_2 < \dots < i_k \leq a.n$ и $1 \leq j_1 < j_2 < \dots < j_k \leq b.n$, что $\forall t : a_{i_t} = b_{j_t}$, $a_{i_t} < a_{i_{t+1}}$ и k максимально.

Формат входного файла

На первой строке целые числа n и m от 1 до 3 000 — длины последовательностей. Вторая строка содержит n целых чисел, задающих первую последовательность. Третья строка содержит m целых чисел, задающих вторую последовательность. Все элементы последовательностей — целые неотрицательные числа, не превосходящие 10^9 .

Формат выходного файла

Выведите одно целое число — длину наибольшей общей возрастающей подпоследовательности.

Система оценки

Подзадача 1 (50 баллов) $n \leq 400$.

Подзадача 2 (50 баллов) $n \leq 3\,000$.

Пример

stdin	stdout
6 5 1 2 1 2 1 3 2 1 3 2 1	2

Разбор задачи О. Наибольшая общая возрастающая

$f_{i,j}$ — максимальная длина общей подпоследовательности, где i — конец подпоследовательности a , и j — любая позиция большая конца в b .
Переходы:

1. $(i, j) \rightarrow (i, j + 1)$ Означает, что мы пропускаем элемент номер j последовательности b .
2. $\text{if } b[j] > a[i] : (i, j) \rightarrow (\text{next}(i, b_j), j + 1)$ Означает, что пару $(\text{next}(i, b_j), j)$ мы взяли в общую подпоследовательность. Здесь $\text{next}(i, b_j)$ — ближайший справа от i элемент, равный b_j , в последовательности a . Чтобы реализовать функцию `next`, удобно сжать координаты. После этого $1 \leq a_i, b_i \leq 2n$ и можно создать для каждого число от 1 до $2n$ упорядоченный список позиций в последовательности a .

Задача Р. k-Рюкзак

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	16 Мб

Вы грабите сокровищницу. У вас есть один рюкзак, а в сокровищнице есть много ценных вещей. Некоторые вещи встречаются несколько раз. Например, обычных золотых монет может быть порядка миллиарда штук. Каждая вещь обладает ценностью и весом. Никакую вещь нельзя разделять на более мелкие части (тогда ценность вещи сразу падает до нуля). Ваша задача — определить максимальную суммарную ценность вещей, которые можно унести из сокровищницы за один заход.

Формат входного файла

На первой строке натуральное число n — количество групп вещей и натуральное число s — размер рюкзака. Следующие n строк содержат тройки натуральных чисел w_i, c_i, k_i , что означает, что в очередную группу входят k_i одинаковых вещей, каждая веса w_i и стоимости c_i . $1 \leq w_i, c_i \leq 10^9$.

Формат выходного файла

Одно целое число — максимальная суммарная ценность вещей, которые можно унести из сокровищницы за один заход.

Система оценки

Подзадача 1 (30 баллов) $n \leq 100$ $s \leq 10^4$ $k_i = 1$
Подзадача 2 (35 баллов) $n \leq 100$ $s \leq 10^4$ $1 \leq k_i \leq 10^9$
Подзадача 3 (35 баллов) $n \leq 300$ $s \leq 10^5$ $1 \leq k_i \leq 10^9$

Пример

stdin	stdout
2 100	339
2 1 100	
7 100 3	

Примечание

В примере выгодно взять все 3 вещи из второй группы. Их суммарный вес — 21, суммарная стоимость — 300. После этого нужно взять 39 вещей из первой группы, их суммарный вес — 78, суммарная стоимость — 39. Больше в рюкзак ничего не влезет.

Разбор задачи Р. k-Рюкзак

Предполагаем, что $k_i = 1$, получаем решение за $O(ns)$ времени с $O(s)$ памяти. Пусть текущий слой динамики f_0 , следующий слой динамики f_1 . Научимся делать переход для группы вещей (w_i, c_i, k_i) за $O(s)$. Можно разбить переход на w_i частей по остаткам по модулю w_i : $0 \dots w_i - 1$. При фиксированном остатке r мы хотим посчитать $f_1[r], f_1[r+w_i], f_1[r+2w_i], \dots$

$$f_1[r + tw_i] = \min_{j=0..k} (f_0[r + (t - k)w_i] + kc_i)$$

Левая и правая граница возрастают, поэтому минимум на отрезке легко считается за $O(1)$.

Задача Q. Разбиения на слагаемые

Вход: stdin
 Выход: stdout
 Ограничение по времени: 0.5 с
 Ограничение по памяти: 16 Мб

Даны целые числа N и K . Нужно сказать, сколько способов разбить число N на ровно K различных положительных целых упорядоченных слагаемых.

Подзадача 1 (33 балла) $1 \leq N, K \leq 500$.
Система оценки Подзадача 2 (33 балла) $1 \leq N, K \leq 10\,000$.
Подзадача 3 (34 балла) $1 \leq N, K \leq 50\,000$.

Формат входного файла

На первой строке через пробел записаны целые числа N и K .

Формат выходного файла

Единственное число — ответ на задачу по модулю $10^9 + 7$.

Пример

stdin	stdout
10 3	4

Пояснение

Способы разбить число 10 на три различных целых упорядоченных слагаемых таковы: $1 + 2 + 7$, $1 + 3 + 6$, $1 + 4 + 5$, $2 + 3 + 5$.

Разбор задачи Q. Разбиение на слагаемые

См. лекцию. «Часть 7».

Задача R. Волшебный лес

Вход: stdin
Выход: stdout
Ограничение по времени: 0.3 с (0.4 для Java)
Ограничение по памяти: 256 Мб

Жила-была матрица $n \times n$. В каждой клетке стояло целое неотрицательное число. Но пришли злые программисты и остались от матрицы только суммы в строках и столбцах.

Ваша задача — восстановить исходную матрицу. Если ответов несколько, минимизируйте максимальное из чисел в матрице. Если ответ все еще не однозначен, выведите любой возможный.

Формат входного файла

В первой строке целое число n ($2 \leq n \leq 1000$). Во второй строке суммы в строках матрицы. В третьей строке суммы в столбцах матрицы. Все суммы — целые числа от 0 до 10^6 .

Формат выходного файла

Если матрица с указанными ограничениями существует, выведите YES и собственно матрицу. Иначе выведите NO.

Система оценки

Подзадача 1 (50 баллов) $n \leq 320$

Подзадача 2 (50 баллов) $n \leq 1000$

Примеры

stdin	stdout
2 1 3 2 2	YES 0 1 2 1
2 1 3 2 4	NO

Разбор задачи R. Волшебный лес

Решение — жадность.

Очевидный критерий существования ответа: сумма сумм в строках равна сумме сумм в столбцах. Далее предполагаем, что эти суммы равны, тогда ответ существует. Конструктивное доказательство: есть строка с положительной суммой, столбец с положительной суммой, увеличиваем значение в матрице на пересечении данных строк и столбца на 1, уменьшаем сумму в строке на 1 и сумму в столбце на 1, повторяем процесс, пока все суммы не обнулятся.

Делаем бинарный поиск по ответу. Внутри проверяем, что ответ меньше либо равен x , следующей процедурой:

```

1 Check(x)
2   Отсортировали столбцы в порядке убывания суммы
3   sum_col = 0
4   for (i = 1; i <= n; i++)
5       sum_col += сумма в i-м столбце
6   sum_row = 0
7   for (j = 1; j <= n; j++)
8       sum_row += min(row[j], i * x)
9   if sum_row < sum_col
10       return false
11   return true

```

После того, как мы знаем, что ответ равен x , нужно расставить числа в матрице. Перебираем строки. Расставляем числа в i -й строке. Когда в некоторую ячейку матрицы ставим число, уменьшаем сумму в соответствующем

столбце. Нужно пытаться в первую очередь уменьшить столбцы с большой суммой. Определяем бинарным поиском максимальное y , что все столбцы можно уменьшить или на x , или до уровня y . После этого уменьшаем все столбцы до уровня y и еще какие-то столбцы до уровня $y - 1$.

Все в сумме работает за $O(n^2 \log M)$. Вторую часть (восстановление матрицы) при желании можно реализовать без использования бинарного поиска за $O(n^2)$.