

Министерство образования и науки Украины



Зимняя школа по программированию

Харьков, ХНУРЭ
2014

Оглавление

День первый. Контест Евгения Капуна	8
Об авторе	8
Теоретический материал. Теория вероятностей	9
Задачи и разборы	13
Задача А. Трудный путь (Юниорская лига)	13
Задача В. Случайное совпадение (Юниорская лига)	15
Задача С. Полный набор (Юниорская лига)	17
Задача D. Питание	18
Задача Е. Секретный код	20
Задача F. Dura Lex	22
Задача G. Путь к знаниям	24
Задача Н. Гранит науки	25
Задача I. Вероятный диагноз	27
Задача J. Зоологический эксперимент	30
Задача K. Игра (Высшая лига)	31
Задача L. Опасная игра (Высшая лига)	33
Задача M. Хеш-таблица (Высшая лига)	34
День второй. Контест Сергея Нагина	37
Об авторе	37
Теоретический материал. Поиск состояний в задачах на динамическое программирование	37
Задачи и разборы	40
Задача A. Сережа и уменьшения (Юниорская лига)	40
Задача B. Сережа и шаблон (Юниорская лига)	41
Задача C. Сережа и массив (Юниорская лига)	43
Задача D. Сережа и вода	44
Задача E. Сережа и игра	45
Задача F. Сережа и столбики	46
Задача G. Сережа и деление	47
Задача H. Сережа и последовательность	48
Задача I. Сережа и экзамен	49
Задача J. Сережа и уменьшения (Высшая лига)	50
Задача K. Сережа и шаблон (Высшая лига)	52
Задача L. Сережа и массив (Высшая лига)	53
День третий. Контест Алексея Шмелева	55
Об авторе	55
Теоретический материал. Лемма Бёрнсайда и Теорема Пойа	56
Задачи и разборы	61
Задача A. Who Calls the Crystal Maiden? (Юниорская лига)	61

Задача B. World of Dota: Cross (Юниорская лига)	63
Задача C. Warlock (Юниорская лига)	64
Задача D. Wild Card: Subway	66
Задача E. Wild Card: Bus	68
Задача F. Windrunner at Your Service	70
Задача G. What is the Answer?	72
Задача H. We Admit No Defeat	73
Задача I. Wex-Wex-Wex	76
Задача J. Warlock-2 (Высшая лига)	79
Задача K. World of Dota: Cross 2 (Высшая лига)	81
Задача L. Wild Card: Plane (Высшая лига)	83
День четвёртый. КонTEST Ярослава Твердохлеба и Романа Едемского	86
Об авторах...	86
Теоретический материал. Оптимизация динамического програмирования за счет использования свойств линейных функций и алгоритма Грэхема	87
Задачи и разборы	93
Задача A. Счастливые пары (Юниорская лига)	93
Задача B. Лазер (Юниорская лига)	94
Задача C. Новогодние подарки (Юниорская лига)	95
Задача D. Депутаты на дереве (Юниорская лига)	97
Задача E. LinearMapReduce	98
Задача F. Круги и деревья	100
Задача G. Петя и игра	101
Задача H. Петя и массивы	104
Задача I. Петя и массив 2	105
Задача J. Петя и прямоугольники	107
Задача K. Игровелоперы	109
Задача L. Путешествие (Высшая лига)	111
Задача M. Петя и среднее (Высшая лига)	113
Задача N. Депутаты на дереве (Высшая лига)	116
День пятый. КонTEST Романа Андреева	119
Об авторе...	119
Теоретический материал. Лекция про пересечение полуплоскостей	120
Задачи и разборы	125
Задача A. Гарри Поттер и три заклинания	125
Задача B. Антисортировка	126
Задача C. Астрид и квадраты	127
Задача D. Берт и землеройки	129
Задача E. Марсианская архитектура	131
Задача F. Камилла и язык программирования WR	132

Задача G. Ворчливые коровы	134
Задача Н. Давид и осёл	136
Задача I. Эмма и соты	138
Задача J. Побег	139
Задача K. Петя и Java	141
Задача L. Цепная дробь	143
Задача M. Шестерёнки	145
Задача N. Задача на НОК	146
Задача O. Сон Студента	147
Задача P. Постройка пирамиды	149
Задача Q. Треугольная комната	150
Задача R. SMS	151
Задача S. Неквадраты	153
Задача T. Перестановка цифр	154
Задача U. Пристрастный учитель	155
Задача V. Поезда	157
Задача W. Ловушки	158
Задача X. Магия вуду	159
Задача на самое короткое решение. Суперминимум	161
День шестой. КонTEST Натальи Бондаренко	162
Об авторе...	162
Теоретический материал. Об одной комбинаторной игре	162
Задачи и разборы	170
Задача A. Игра с числом	170
Задача B. Отношение чисел Фибоначчи (простая версия)	171
Задача C. Отношение чисел Фибоначчи (сложная версия)	173
Задача D. Обобщенная последовательность Фибоначчи	174
Задача E. Сложение в фибоначиевой системе счисления	176
Задача F. Игра с монетами	178
Задача G. Цзяньшицы с тремя кучками	179
Задача H. Спички детям не игрушки	180
Задача I. Ферзя в угол!	182
Задача J. (p, q) -коня в угол!	183
Задача K. Игра	185
Задача L. Игра со строкой	191
День седьмой. КонTEST Сергея Копелиовича	193
Об авторе...	193
Теоретический материал. Перебор и NP-полные задачи	194
Задачи и разборы	199
Задача A. Трисочетание	199
Задача B. Множество множеств	200

Задача С. Длинная дорога	202
Задача Е. Корни	203
Задача F. Умножение многочленов	204
Задача G. Деление многочленов	205
Задача Н. Ребра добавляются, граф растет	206
Задача I. Сумма всего подряд	207
Задача J. Все клики	209
Задача К. Макс клика	210
Задача L. Учимся красить	212
Задача М. Проще всего красить в три цвета	213
Задача N. SAT USAT	215
Задача О. Почти двудольный зверь	216
День восьмой. Контест Akai	219
Об авторе...	219
Теоретический материал. Поиск минимального элемента в стеке, в очере- ди и в скользящем окне в массиве	219
Задачи и разборы	222
Задача A. Anti-Lines (высшая лига)	222
Задача B. Big Fellow (высшая лига)	225
Задача C. Concave Polygon (высшая лига)	227
Задача D. Dead Points (высшая лига)	229
Задача E. Equilateral Polygon	230
Задача F. Flawless Numbers	231
Задача G. Grep	233
Задача H. Hash It!	234
Задача I. Interactive Problem 2	236
Задача J. Jolly Dolls	238
Задача K. Block Shuffling (юниорская лига)	240
Задача L. Digit Permutation (юниорская лига)	241
Задача M. Mortal Points (юниорская лига)	242
Задача N. Non-convex Polygon (юниорская лига)	244
День девятый. Контест Виталия Неспирного	246
Об авторе...	246
Теоретический материал. Построение пересечения полуплоскостей	247
Задачи и разборы	252
Задача А. Максимальная разность (высшая лига)	252
Задача В. Максимальная разность (юниорская лига)	253
Задача С. Построение многоугольника (высшая лига)	254
Задача D. Построение треугольника (юниорская лига)	255
Задача Е. Красивые узоры (высшая лига)	260
Задача F. Красивые узоры (юниорская лига)	261

Задача G. Стока без повторений (высшая лига)	264
Задача H. Стока без повторений (юниорская лига)	265
Задача I. Сбор бобов (высшая лига)	268
Задача J. Сбор бобов (юниорская лига)	269
Задача K. Обратный сбор бобов (высшая лига)	271
Задача L. Обратный сбор бобов (юниорская лига)	272
Задача M. Уравнение (высшая лига)	274
Задача N. Уравнение (юниорская лига)	275
Задача O. Соревнование (высшая лига)	276
Задача P. Соревнование (юниорская лига)	278
Задача Q. Вписанная окружность (высшая лига)	280
Задача R. Вписанная окружность (юниорская лига)	281
Задача S. Разделяющая прямая (высшая лига)	284
Задача T. Разделяющая прямая (юниорская лига)	285
Задача U. Стингангуляция (высшая лига)	287
Задача V. Стингангуляция (юниорская лига)	288
Задача W. Построение куба (высшая лига)	290
Задача X. Построение квадрата (юниорская лига)	291

День первый (15.02.2014 г.) Контест Евгения Капуна

Об авторе...

Капун Евгений Дмитриевич, родился 2 октября 1989 года в Санкт-Петербурге, в 2006 году закончил ФТШ, а в 2012 году стал магистром кафедры компьютерных технологий факультета информационных технологий и программирования НИУ ИТМО.



Основные достижения:

- 1 место на финале ACM ICPC 2009 и 2012 (в составе команды ИТМО).
- 4 место на ACM NEERC 2009 и 2010, 1 место на ACM NEERC 2011 (в составе команды ИТМО).
- 2 место на финале Russian Code Cup 2011, 5 место на финале Russian Code Cup 2012.
- 2 место на финале Яндекс.Алгоритма 2013.
- 1 место на Чемпионате Урала 2009, 2010 и 2012 (в составе команды ИТМО).
- 1 место на Открытой Всесибирской Олимпиаде по программированию имени И. В. Потоссина 2008, 2009 и 2010 (в составе команды ИТМО).
- 2 место на турнире по программированию ICL 2011 (в составе команды ИТМО).
- 3 место на чемпионате по программированию КРОК-2013.
- 4 место в V открытом кубке имени Е. В. Панкратьева по программированию и 3 место в VII, VIII, X и XI кубках (в составе команды ИТМО), 2 место в XIII кубке (в составе команды Petr Team).

Теоретический материал. Теория вероятностей

Определения

Теория вероятностей рассматривает некоторый случайный процесс (или совокупность случайных процессов), называемый опытом. Возможные результаты этого опыта называются исходами. Множество исходов (обычно его обозначают буквой Ω) — множество, на котором определена мера, такая, что мера всего множества равна 1. Если говорить простым языком, то мера — это функция, которая сопоставляет некоторый неотрицательный вес каждому элементу множества, а за меру подмножества принимается сумма мер его элементов. Мера каждого исхода называется его вероятностью и указывает, насколько ожидаем тот или иной исход.

Событие — подмножество множества исходов. Вероятность события — его вес. Вероятность события A обозначается $P(A)$.

Пример

Пусть множество исходов состоит из двух элементов, и каждому из них соответствует вероятность $\frac{1}{2}$. Тогда существует четыре различных события: пустое подмножество, два подмножества из одного элемента и всё множество. В элементарной теории вероятностей часто встречаются конечные множества исходов, в которых все исходы равновероятны.

Сведение задач по теории вероятностей к задачам по комбинаторике

Часто задачи по теории вероятностей можно свести к задачам по комбинаторике. Обычно в этих задачах требуется найти вероятность того, что результат некоторого случайного выбора, имеющего несколько равновероятных исходов, принадлежит некоторому подмножеству. В этом случае ответом будет размер подмножества, делённый на размер всего множества возможных исходов.

Пример. Задача: найти вероятность того, что 10 наугад выбранных чисел от 1 до 10 будут различными. Решение: всего существует 10^{10} способов выбрать 10 чисел от 1 до 10, и при выборе наугад вероятности всех этих способов совпадают. Из них, в $10!$ способах все числа будут различными. Ответ: $\frac{10!}{10^{10}}$.

Вероятностная динамика

При решении задач по теории вероятностей часто используется динамика, в которой значением является вероятность (или несколько вероятностей). Будем называть такую динамику вероятностной. Обычно в таких задачах переходы между состояниями необратимы (пройдя состояние, нельзя в него вернуться), а в итоге происходит переход или в “хорошее” состояние (вероятность 1), или в “плохое” (вероятность 0).

Такие задачи удобно решать с конца, при этом значением динамики в некотором состоянии является некоторая функция $P(x)$ (вероятность события x), если начинать из этого состояния. Чтобы посчитать это значение для некоторого состояния, нужно знать его для всех состояний, в которые из него можно перейти. Формула обычно имеет вид $D_i = \sum_j P_{ij} D_j$, где D_i — значения динамики, а P_{ij} — вероятности перехода.

Примером использования вероятностной динамики является такая игра: в каждом раунде первый игрок выигрывает с вероятностью p , а второй — с вероятностью $q = 1 - p$. Если выигрывает первый игрок, то второй отдаёт ему одну монету, если же выигрывает второй, то монету отдаёт первый игрок. Игра заканчивается, когда у одного из игроков не остаётся денег. Если у первого игрока изначально n_1 монет, а у второго n_2 , то вероятность выигрыша первого игрока равна $\frac{p^{n_1+n_2} - p^{n_2} q^{n_1}}{p^{n_1+n_2} - q^{n_1+n_2}}$, а второго — $\frac{q^{n_1+n_2} - p^{n_2} q^{n_1}}{q^{n_1+n_2} - p^{n_1+n_2}}$. Этот результат известен как теорема Гюйгенса (кстати, если $p = q = \frac{1}{2}$, то всё намного проще — вероятности равны $\frac{n_1}{n_1+n_2}$ и $\frac{n_2}{n_1+n_2}$ соответственно).

Чтобы доказать это, можно воспользоваться вероятностной динамикой, однако, в отличии от обычного случая, здесь переходы обратимы, поэтому просто вычислить вероятности по порядку не получится. Вместо этого вероятностная динамика даёт систему линейных уравнений, которые можно решить. Таким образом, если переходы необратимы, значения можно просто вычислять друг за другом, а если обратимы, то придётся решать систему уравнений.

Условные вероятности. Формула Байеса

Условная вероятность — это вероятность того, что произошло некоторое событие, но только если рассматривать исходы, при которых произошло какое-то другое событие. Формально $P(A|B)$ — условная вероятность того, что событие A произошло при условии события B , — равна $\frac{P(A \cap B)}{P(B)}$. Чтобы это определение было осмысленным, необходимо, чтобы вероятность события B была ненулевой.

Как следствие,

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A).$$

Если $\bigsqcup_i A_i = \Omega$ — разбиение множества исходов на события A_i , то $P(B) = \sum_i P(A_i \cap B) = \sum_i P(B|A_i)P(A_i)$. Отсюда следует, что $P(A_i|B) = \frac{P(A_i \cap B)}{P(B)} = \frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_j)P(A_j)}$. Этот результат известен как формула Байеса и очень полезен тем, что позволяет “разворачивать” условные вероятности.

Независимые события

События A и B называются независимыми, если $P(A \cap B) = P(A)P(B)$. Если $P(B) > 0$, то это то же самое, что $P(A|B) = P(A)$. Это значит, что зна-

ние того, что произошло некоторое событие B , не влияет на наше представление о вероятности события A . Таким образом, эти события в некотором смысле никак не влияют друг на друга. Вероятностную независимость можно определить и для большего числа событий, причём независимость множества событий — более сильное утверждение, чем попарная независимость всех этих событий (и даже чем независимость всех собственных подмножеств).

Случайные величины. Математическое ожидание

Случайная величина — это величина, которая может принимать различные значения в зависимости от исхода. Это значит, что каждому возможному значению случайной величины можно сопоставить вероятность. Простейший пример: величина X , которая принимает значение 0 с вероятностью $\frac{1}{2}$ и значение 1 с вероятностью $\frac{1}{2}$. Записывают так: $P(X = 0) = P(X = 1) = \frac{1}{2}$. Соответствие между множеством значений случайной величины и их вероятностями называется распределением случайной величины.

Пример распределения случайной величины — геометрическое распределение. Оно принимает параметр p , и если величина X имеет такое распределение, то $P(X = i) = qp^i$, где $q = 1 - p$, для всех целых $i \geq 0$. Заметим, что это распределение дискретно — величина может принимать только целые значения. Несложно проверить, что сумма вероятностей по всем возможным значениям i равна 1.

Геометрическое распределение часто встречается на практике как распределение числа неудачных попыток перед первой удачной в испытании, в котором вероятность успеха всегда одинаковая (и равна p). Например, если подбрасывать монетку, то число орлов до первой решки имеет геометрическое распределение с параметром $p = \frac{1}{2}$.

Случайные величины можно складывать, умножать и вычислять результат других математических действий над ними. На значение функции от нескольких величин может влиять вероятность определённых комбинаций их значений. Например, если две случайные величины всегда равны, то их разность всегда равна нулю. Две величины X и Y называются независимыми, если события $X = x$ и $Y = y$ независимы для всех x и y .

Математическое ожидание случайной величины — в некотором смысле её среднее значение. Формально $E(X) = \sum_x xP(X = x)$, где x пробегает все возможные значения X . Математическое ожидание суммы случайных величин равно сумме их математических ожиданий, а математическое ожидание произведения случайной величины и константы равно произведению математического ожидания этой же величины и этой же константы: $E(X + Y) = E(X) + E(Y)$ и $E(cX) = cE(X)$. Если случайные величины независимы, то произведение их математических ожиданий равно математи-

ческому ожиданию их произведения, но в общем случае это неверно.

Вероятностная динамика для математического ожидания

Во многих задачах требуется найти математическое ожидание числа шагов до наступления некоторого события. В таких задачах можно использовать динамику, в которой значением является искомое математическое ожидание при условии, что в качестве начального используется текущее состояние. Формула пересчета обычно выглядит так: $D_i = 1 + \sum_j P_{ij} D_j$. Как и в обычной вероятностной динамике, значения вычисляются с конца, а если переходы обратимы, то приходится решать систему линейных уравнений.

Пример: лягушка начинает с точки с координатой 0 и за один шаг из точки с координатой i с равной вероятностью прыгает в точку с координатой $i + 1$ или $i - 1$. Найти, за сколько шагов (в среднем) она попадёт в точку, координата которой не меньше n . Решение: состояние динамики — координата лягушки, значение — искомая величина, но при условии, что лягушка начинает путь из текущей точки. Формулы: для $i \geq n$ $D_i = 0$, иначе $D_i = 1 + \frac{D_{i+1} + D_{i-1}}{2}$.

Кроме вариантов с необратимыми и обратимыми переходами есть промежуточный вариант, когда переходы необратимы, за исключением возможных переходов из некоторых состояний в себя. В этом случае нужно действовать так же, как и с необратимыми переходами, но на каждом шаге решать линейное уравнение с одной неизвестной.

Цепи Маркова

Цепь Маркова — случайный процесс, в котором очередное состояние зависит только от предыдущего состояния и, возможно, от порядкового номера. Если зависимости от номера нет, то цепь называется однородной. Именно такие цепи обычно встречаются в задачах. Более того, обычно у них конечное число состояний, а иногда даже можно явно построить матрицу переходов.

Матрица переходов $M = \|p_{ij}\|$, где p_{ij} — вероятность перехода из состояния i в состояние j и для всех $i \sum_j p_{ij} = 1$. Матрица переходов в цепи Маркова всегда является стохастической — все её элементы неотрицательны и сумма элементов в каждой строке равна единице. Также представляет интерес граф переходов — ориентированный граф, в котором есть ребро из вершины i в вершину j , если $p_{ij} > 0$. Любая возможная последовательность состояний цепи Маркова соответствует пути в графе переходов, и, наоборот, любой путь в графе переходов соответствует возможной последовательности состояний в цепи.

Стационарное распределение — распределение состояний цепи, которое не изменяется со временем, то есть если распределение некоторого состояния является стационарным, то все последующие состояния будут распределены

так же. Если вероятности состояний заданы вектором π , то $\pi M = \pi$. Часто, но не всегда, распределение в цепи со временем стремится к стационарному.

Состояния в цепи Маркова можно классифицировать по достижимости. Состояние j достижимо из состояния i , если система может из состояния i перейти в состояние j за конечное число шагов (записывают $i \rightarrow j$). Состояния i и j коммуницируют (записывают $i \leftrightarrow j$), если $i \rightarrow j$ и $j \rightarrow i$. Это отношение порождает классы эквивалентности, называемые неразложимыми классами. Если вся цепь представляет собой один такой класс, то её также называют неразложимой.

Состояние называется периодическим, если, выйдя из него, вернуться можно только через число шагов, кратное некоторому целому числу, большему единицы. В противном случае состояние называют апериодическим. Можно доказать, что в неразложимом классе все состояния или апериодические, или периодические с одинаковым периодом.

Оказывается, если цепь Маркова неразложима и апериодична, то у неё есть единственное стационарное распределение, к которому она будет стремиться из любого начального распределения. Этот результат известен как теорема Фробениуса — Перрона. Если цепь неразложима и периодична, то всё равно распределения состояний, номера которых сравнимы между собой по модулю периода, имеют предел (например, если период равен двум, то распределения чётных и нечётных состояний имеют предел, но эти пределы могут не совпадать). Эти пределы могут зависеть от начального распределения.

Существует несколько приёмов работы с цепями Маркова. Например, можно использовать алгоритм быстрого возведения матрицы в степень, чтобы быстро считать распределение состояний через большое число шагов. Чтобы найти стационарное распределение, нужно решить систему уравнений $\pi M = \pi$ (по одному уравнению для каждой координаты), в которой вместо одного из уравнений вставлено уравнение $\sum p_i = 1$ (заменить можно любое из уравнений).

Задачи и разборы

Задача А. Трудный путь (Юниорская лига)

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

Вася хорошо выпил и теперь, когда он добрался до своей улицы, он полностью потерял чувство направления. Поскольку он не помнит, с какой стороны

его дом, он выбирает направление наобум. Более того, на каждом перекрёстке он с вероятностью 50% продолжает идти вперёд, а иначе разворачивается и идёт назад. Он настолько потерял связь с реальностью, что может даже пройти мимо своего дома и не заметить этого!

Пройдя N кварталов, Вася засыпает прямо на улице. Проснувшись, он задаётся вопросом: какой у него был шанс заснуть рядом с домом? Ведь от перекрёстка, от которого он начал свой путь, до перекрёстка рядом с домом Васи всего M кварталов. Помогите ему.

Ограничения

$$\begin{aligned}1 &\leq N \leq 1000 \\0 &\leq M \leq 1000\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит два целых числа: N и M .

Формат выходного файла

Выведите единственное число — вероятность Васи заснуть на перекрёстке рядом со своим домом. Выведите ответ с абсолютной погрешностью не более 10^{-7} .

Пример

stdin	stdout
1 1	0.5
10 20	0.0
1000 100	0.0001694

Разбор задачи А. Трудный путь

Будем представлять путь Васи как строку из нулей и единиц, где ноль обозначает квартал, пройденный в одном направлении, а единица — в другом. Тогда, если $N = 1$, то очевидно, что два возможных пути — 0 и 1 — равновероятны. На каждом следующем шаге к пути с равной вероятностью дописывается 0 и 1, так что после N шагов получается 2^N возможных путей, и все они равновероятны.

Пусть в пути K единиц. Несложно заметить, что точка, где закончится путь, отстоит от начальной на $|N - 2K|$ кварталов, причём $0 \leq K \leq N$. Значит, если $M > N$ или M и N имеют разную чётность, ответ равен нулю, так как такой точке не соответствует ни один путь. Иначе $K = \frac{N+M}{2}$ (будем считать, что

единица — это квартал, пройденный в направлении дома) и ответ равен $\frac{C_N^K}{2^N}$ — числу допустимых путей, деленному на общее число путей.

Значение C_N^K легко вычисляется с помощью факториалов. Однако, если производить вычисления с использованием типа `double`, то может произойти переполнение. Чтобы этого избежать, можно использовать тип `long double` или вместо самих значений производить операции над их логарифмами.

Задача В. Случайное совпадение (Юниорская лига)

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

Вася скучно, и он от нечего делать взял N кубиков, пронумерованных числами от 1 до N , перемешал их и расставил в случайном порядке. После этого он несколько раз перемешал кубики и расставил их в том порядке, в котором они оказались. После каждого перемешивания он смотрел, не получилась ли та же самая перестановка, что и перестановка, получившаяся после первого перемешивания. Но кубиков было много, искомая перестановка всё никак не получалась, и Вася опять стало скучно. Но Вася подумал и решил получить искомую перестановку другим способом. Он расставил кубики по порядку от 1 до N и решил выбрать часть кубиков, перемешать их и вернуть на те же места, но, возможно, в другом порядке. Он решил повторять эти действия до тех пор, пока не получится перестановка, которую он запомнил в самом начале, но перед этим ему хочется узнать, через какое время, в среднем, он сможет этого добиться? Вася хочет получить эту перестановку как можно быстрее и выбирает кубики, которые он будет перемешивать, соответственно.

Ограничения

$$1 \leq N \leq 1000$$

Формат входного файла

Первая строка входного файла содержит целое число N .

Вторая строка содержит N целых чисел от 1 до N — перестановка, которую хочет получить Вася.

Формат выходного файла

Выведите единственное число — среднее количество перемешиваний, после которых Вася получит искомую перестановку.

Пример

stdin	stdout
3 2 1 3	2.0
3 3 1 2	3.0

Пояснение

В первом примере Вася каждый раз выбирает для перемешивания первые два кубика. С вероятностью $\frac{1}{2}$ после перемешивания кубики окажутся переставленными, и на этом процесс закончится. В среднем это произойдёт после двух перемешиваний.

Разбор задачи В. Случайное совпадение

Пусть текущая перестановка равна $\{P_i\}_{i=1}^N$, и для очередного перемешивания Вася выбрал множество из n кубиков, находящихся в позициях $\{i_k\}_{k=1}^n$. Поскольку все перестановки кубиков равновероятны, после перемешивания каждый из n перемешиваемых кубиков с вероятностью $\frac{1}{n}$ окажется на каждой из n позиций. Из этих позиций в лучшем случае одна будет совпадать с позицией этого кубика в искомой перестановке. Будем считать, что $X_k = 1$, если кубик из позиции i_k после перемешивания попал на позицию в искомой перестановке, иначе $X_k = 0$. В этом случае у тех кубиков, у которых позиция в искомой перестановке находится среди позиций перемешиваемых кубиков, $E(X_k) = \frac{1}{n}$, у остальных же кубиков $E(X_k) = 0$. Но $\sum_{k=1}^n X_k$ — это число кубиков (из перемешиваемых), которые после перемешивания оказались на целевых позициях, и $E(\sum_{k=1}^n X_k) = \sum_{k=1}^n E(X_k) \leq 1$. Значит, из всех перемешиваемых кубиков в среднем не более одного окажется на целевой позиции, а равенство достигается, когда для каждого из перемешиваемых кубиков его целевая позиция также входит в множество позиций перемешиваемых кубиков.

Значит, за каждое перемешивание число кубиков, стоящих на своих местах, можно увеличить, в среднем, не более чем на один. Как увеличить ровно на один? Оказывается, для этого достаточно каждый раз перемешивать ровно те кубики, которые стоят не на своих местах. До перемешивания среди них ноль кубиков стоят на своих местах, кроме того, для каждого из кубиков в перемешиваемое множество входит и кубик на позиции, где он должен стоять (так как он тоже стоит не на месте). Значит, после перемешивания в среднем ровно один из этих кубиков будет стоять на своём месте, то есть на один больше, чем до перемешивания.

Докажем, что если n кубиков стоят не на своих местах, то ответ (назовём его A_n) равен n . Доказывать будем по индукции. Если $n = 0$, то все кубики стоят на своих местах и ничего делать не надо. Иначе перемешаем те кубики, которые стоят не на своих местах. Пусть после этого m кубиков стоят не на своих местах, причём $0 \leq m \leq n$, так как кубики, уже стоящие на своих местах, при перемешивании не затронуты. Тогда, согласно доказанному ранее, $E(m) = n - 1$, то есть

$$P(m = 1) + 2P(m = 2) + \dots + nP(m = n) = n - 1.$$

Но тогда

$$A_n = 1 + A_0 P(m = 0) + A_1 P(m = 1) + \dots + A_{n-1} P(m = n-1) + A_n P(m = n).$$

По предположению индукции для всех $m < n$ $A_m = m$, из чего получается, что $A_n = n$. То, что такая стратегия оптимальна, доказывается аналогично: если на очередном шаге сделать выбор по-другому, то A_n получится не меньше, чем при данной стратегии.

Задача С. Полный набор (Юниорская лига)

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

Вася продолжает умирать от скуки. Чтобы хоть как-то развлечься, он взял N кубиков, пронумерованных от 1 до N , перемешал их и взял K из них наобум, после чего записал их номера и вернул их в общую кучу. Затем он повторил эти действия: снова перемешал, снова взял K кубиков, и так далее. И теперь у него возник вопрос: сколько раз нужно так сделать, чтобы каждый кубик был взят хотя бы по одному разу?

Ограничения

$$\begin{aligned}1 &\leq N \leq 1000 \\1 &\leq K \leq N\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит два целых числа: N и K .

Формат выходного файла

Выведите единственное число — среднее количество итераций до того, как каждый кубик будет взят хотя бы по разу. Выведите ответ с относительной погрешностью не более 10^{-7} .

Пример

stdin	stdout
5 1	11.4166667
5 5	1.0

Разбор задачи С. Полный набор

Поскольку все кубики равнозначны, в каждый момент времени нас интересует только то, сколько кубиков было взято хотя бы по одному разу. Будем реализовывать вероятностную динамику, где состояние — количество уже взятых когда-либо кубиков, а значение — среднее количество итераций, начиная с этого момента, до того, как будут взяты все кубики.

Динамика считается с конца. Для N значение динамики равно 0, поскольку все кубики уже взяты. Пусть взято i кубиков, тогда на следующей итерации может быть взято $i + j$ кубиков, где $0 \leq j \leq K$ и $K \leq i + j \leq N$. Вероятность того, что после i будет $i + j$, считается так: всего есть C_N^K способов выбрать K кубиков из N , и все они равновероятны. Из них $C_i^{K-j} C_{N-i}^j$ способов приведут к тому, что на следующей итерации будет взято ровно $i + j$ кубиков. Таким образом, вероятность этого равна $\frac{C_i^{K-j} C_{N-i}^j}{C_N^K}$.

Теперь несложно написать формулу для перехода: если $i < k$, то $D_i = 1 + \frac{\sum_{K-i \leq j \leq \min(N-i, K)} C_i^{K-j} C_{N-i}^j D_{i+j}}{C_N^K}$, если же $i \geq k$, то $D_i = \frac{C_N^K + \sum_{1 \leq j \leq \min(N-i, K)} C_i^{K-j} C_{N-i}^j D_{i+j}}{C_N^K - C_i^K}$ где D_i — значение динамики в состоянии i , а $D_N = 0$. Ответом задачи будет значение D_0 .

Задача D. Питание

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

Вася решил отправиться в путешествие. Сейчас он летит на самолёте, где как раз начинают раздавать обед. Обед бывает двух видов — мясной и рыбный, и бортпроводники спрашивают у каждого из пассажиров, какой вид обеда они предпочитают.

Но Вася знает, что каждого вида обеда в отдельности не хватит на всех пассажиров. Если обеды одного вида заканчиваются, бортпроводники перестают спрашивать пассажиров об их предпочтениях и просто дают им то, что

осталось. Вася очень хочет получить рыбный обед. Он знает, что на самолёт, на котором летят N пассажиров, взято N_1 мясных и N_2 рыбных обедов. Кроме того, он посмотрел, в каком порядке раздают обеды, и заметил, что он I -й по счёту. Также он предусмотрительно раздобыл статистику, из которой узнал, что рыбные обеды предпочитают $P\%$ пассажиров. Теперь ему не терпится узнать: какой у него шанс получить рыбный обед?

Ограничения

$$\begin{aligned}0 &\leq N_1 \leq 1000 \\0 &\leq N_2 \leq 1000 \\1 &\leq N \leq N_1 + N_2 \\1 &\leq I \leq N \\0 &\leq P \leq 100\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит пять целых чисел: N , N_1 , N_2 , I и P .

Формат выходного файла

Выведите единственное число — вероятность Васи получить рыбный обед. Выведите ответ с абсолютной погрешностью не более 10^{-7} .

Пример

stdin	stdout
10 5 5 10 50	0.5
10 8 7 8 60	0.9720064

Разбор задачи D. Питание

Условие задачи специально усложнено тем, что некоторые пассажиры не всегда делают выбор, а также ненужным значением N . Действительно, Вася совершенно неважно, сколько пассажиров после него будут заказывать обед. Более того, будем считать, что все пассажиры (все — это $I - 1$ пассажиров перед Васей, остальные его не интересуют) сделали выбор, просто некоторые из них не имеют шанса его высказать. Теперь заметим, что рыба достанется Васе тогда и только тогда, когда из пассажиров впереди него рыбку выбрали менее чем N_2 пассажира. Действительно, если перед Васей рыбку выбрали хотя бы N_2 пассажира, то она точно закончится и Васе не достанется, иначе рассмотрим два случая: мясо перед Васей закончилось или не закончилось. В первом случае рыба не может закончиться, так как $N_1 + N_2 \geq N$, а во втором

каждому достанется то, что он выбрал, а рыбу хочет менее, чем N_2 пассажира, поэтому Васе она достанется.

Таким образом, осталось найти вероятность того, что из первых $I - 1$ пассажиров рыбу выбрали меньше, чем N_2 . Заметим, что количество пассажиров, выбравших рыбу, подчиняется биномиальному распределению. Если $N_2 \geq I$, то эта вероятность равна 1, иначе нужно просуммировать вероятности для значений от 0 до $N_2 - 1$:

$$P_a = \sum_{i=0}^{N_2-1} P^i (1-P)^{I-1-i} C_{I-1}^i$$

(здесь считаем, что $0 \leq P \leq 1$, P_a — искомая вероятность).

Задача E. Секретный код

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

Вернувшись из путешествия, Вася был неприятно удивлен: на двери подъезда его дома установлен новый кодовый замок. Вася не может попасть к себе домой! Кодовый замок содержит N дисков, каждый из которых может находиться в одном из M положений. Ровно одна комбинация является подходящей. Внимательно осмотрев диски, Вася по отпечаткам пальцев и царапинам определил вероятность каждого из положений для каждого диска. Теперь у Васи есть K попыток подобрать код: если он не успеет, то бдительные соседи вызовут полицию, и Васе придётся долго доказывать, что он не вор, а просто пытается попасть домой. Помогите Васе посчитать максимальную вероятность оказаться дома, а не в полиции.

Ограничения

$$\begin{aligned}1 &\leq N \leq 100 \\1 &\leq M \leq 20 \\1 &\leq K \leq 100 \\0 &\leq P_{ij} \leq 100\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит три целых числа: N , M и K .

Следующие N строк содержат по M целых чисел каждая: j -е число i -й строки (P_{ij}) — вероятность того, что i -й диск в подходящей комбинации находится в положении j . Гарантируется, что $\sum_{j=1}^M P_{ij} = 100$.

Формат выходного файла

Выведите единственное число — вероятность Васи успеть подобрать код. Выведите ответ с абсолютной погрешностью не более 10^{-7} .

Пример

stdin	stdout
2 2 1 50 50 10 90	0.45
3 5 4 10 15 20 25 30 1 2 3 4 90 100 0 0 0 0	0.81

Разбор задачи Е. Секретный код

Очевидно, что оптимальная стратегия для Васи — попробовать K наиболее вероятных комбинаций. Вероятность того, что некоторая комбинация является допустимой, равна произведению соответствующих вероятностей для каждого диска. Как узнать K самых вероятных комбинаций? Заметим, что если комбинация входит в K самых вероятных, то комбинация из первых $N - 1$ дисков тоже входит в K самых вероятных среди комбинаций из первых $N - 1$ дисков: иначе можно заменить комбинацию из первых $N - 1$ дисков на более вероятную, что увеличит вероятность комбинации из N дисков, таким образом, чтобы она не совпала ни с одной из более вероятных комбинаций. Аналогично, комбинация из первых $N - 2$ дисков также входит в K самых вероятных, как и комбинация из первых $N - 3$, и т. д.

Чтобы посчитать ответ, начнём с комбинации из нуля дисков. Такая комбинация ровно одна. Теперь на каждом шаге будем дописывать ко всем комбинациям все возможные значения следующей цифры, после чего оставлять только K наиболее вероятных. Если для поддержания K наиболее вероятных комбинаций использовать биномиальную кучу, то просеивание KM комбинаций может быть сделано за время $O(KM \log K)$. Всего нужно сделать N шагов, что делается за время $O(NKM \log K)$. Осталось только просуммировать вероятности.

Задача F. Dura Lex

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

Вася обожает заказывать новые гаджеты из-за границы. К сожалению, для Васи, недавно ввели новые правила таможенного контроля, согласно которым для получения каждого гаджета Васе нужно получить N справок. Получить каждую справку непросто — чтобы получить i -ю справку, нужно отстоять в очереди D_i дней, причём одновременно можно стоять в очереди не более чем за одной справкой. И что самое обидное, в выдаче i -й справки отказывают с вероятностью $P_i\%$, причём совершенно случайно, и более того, если в выдаче справки отказано, то все предыдущие справки автоматически аннулируются, и всё приходится начинать сначала. Хоть одно радует — справки можно получать в любом порядке.

Вася хочет получить новый гаджет во что бы то ни стало. Он будет пытаться собрать все справки, пока не добьётся успеха. Помогите ему выбрать порядок получения справок таким образом, чтобы минимизировать среднее время, за которое он соберёт их все.

Ограничения

$$\begin{aligned}1 &\leq N \leq 10^5 \\0 &\leq D_i \leq 1000 \\0 &\leq P_i < 100\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит целое число N .

Следующие N строк содержат по два целых числа каждая: D_i и P_i .

Формат выходного файла

Выведите N целых чисел от 1 до N — искомую перестановку. Если оптимальных перестановок несколько, выведите лексикографически минимальную.

Пример

stdin	stdout
3 1 10 10 10 5 90	3 1 2
3 2 20 3 30 4 40	1 2 3

Разбор задачи F. Dura Lex

Пусть Вася пытается получить i -ю справку. Вероятность получить её с первого раза равна $1 - P_i$, со второго — $P_i(1 - P_i)$ (нужно, чтобы отказали в первый раз, но не во второй), с третьего — $P_i^2(1 - P_i)$ и т. д. Несложно заметить, что количество попыток подчиняется геометрическому распределению и среднее количество попыток до успешного получения справки равно $\frac{1}{1 - P_i}$.

Пусть Вася пытается получить i -ю справку, а затем j -ю. Ему нужно в среднем $\frac{1}{1 - P_i}$ попыток, чтобы получить i -ю справку, но каждое получение i -й справки даёт ему всего одну попытку для получения j -й справки, а таких попыток нужно в среднем $\frac{1}{1 - P_j}$, так что всего попыток будет $\frac{1}{(1 - P_i)(1 - P_j)}$. В каждой из этих попыток Вася попытается получить i -ю справку, а в $\frac{1}{1 - P_j}$ из них — ещё и j -ю, что в сумме занимает, в среднем, $\frac{D_i}{(1 - P_i)(1 - P_j)} + \frac{D_j}{1 - P_j}$ дней.

Пусть Вася пытается получить i_1 -ю справку, затем i_2 -ю, и так далее, затем i_N . Применяя рассуждения, аналогичные выше, можно узнать, что это в среднем это займёт $\frac{D_{i_1}}{(1 - P_{i_1})(1 - P_{i_2})\dots(1 - P_{i_N})} + \frac{D_{i_2}}{(1 - P_{i_2})\dots(1 - P_{i_N})} + \dots + \frac{D_{i_N}}{1 - P_{i_N}}$ дней. Если переставить две соседние справки — i_j -ю и i_{j+1} -ю — то это время увеличится на $\frac{D_{i_{j+1}}P_{i_j} - D_{i_j}P_{i_{j+1}}}{(1 - P_{i_j})\dots(1 - P_{i_N})}$. Соответственно, нам выгодно делать такую замену, если $D_{i_{j+1}}P_{i_j} < D_{i_j}P_{i_{j+1}}$. Можно доказать, что это отношение транзитивно и что такую замену выгодно делать не только для соседних справок.

Соответственно, решение — отсортировать справки, сравнивая их способом, указанным выше. Чтобы получить лексикографически минимальную перестановку, эквивалентные справки нужно выводить по порядку. Небольшая проблема состоит в том, что справки, для которых $D_i = P_i = 0$, не влияют на результат и могут быть вставлены в любое место списка. Предлагается выносить их в отдельный список и в конце объединять с остальными.

Задача G. Путь к знаниям

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

Вася учится в университете и каждый день ходит туда пешком. Город, в котором живёт Вася, представляет собой неориентированный граф. Вася решил использовать научный подход в выборе дороги, поэтому он изучил карту города и нашёл все кратчайшие маршруты от дома до университета. Теперь каждый раз, когда Вася идёт в университет или обратно, он выбирает один из маршрутов, причём каждый маршрут выбирается с равной вероятностью.

Через несколько дней Вася заметил, что через некоторые перекрёстки он ходит чаще, чем через другие. Он решил посчитать, сколько раз в день он в среднем проходит через каждый перекрёсток. Но, поскольку он занят учёбой, он поручил это сделать вам.

Ограничения

$$\begin{aligned}1 &\leq N \leq 10^5 \\0 &\leq M \leq 10^5 \\1 &\leq A_i, B_i \leq N \\1 &\leq L_i \leq 10000\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит два целых числа N и M — количество перекрёстков и дорог в городе, где живёт Вася.

Каждая из следующих M строк соответствует одной улице и содержит три целых числа A_i , B_i и L_i — номера перекрёстков, которые соединяет улица, и её длину в километрах.

Дом Васи находится рядом с первым перекрёстком, а университет — рядом с N -м. Гарантируется, что от дома Васи можно дойти до университета по дорогам.

Формат выходного файла

Выведите N чисел — среднее количество проходов в день через перекрёстки с первого по N -й. Выводите числа с абсолютной погрешностью не более 10^{-7} . Не забывайте, что Вася проходит по городу два раза в день — в университет и обратно.

Пример

stdin	stdout
3 3 1 2 1 2 3 1 1 3 1	2.0 0.0 2.0
4 4 1 2 1 1 3 1 2 4 1 3 4 1	2.0 1.0 1.0 2.0

Разбор задачи G. Путь к знаниям

Эта задача не столько на теорию вероятностей, сколько на умение вычислять количество кратчайших путей, проходящих через некоторую вершину. Вначале нужно посчитать расстояние до каждой из вершин от первой и N -й вершины. Затем, если сумма этих расстояний для какой-то вершины больше, чем расстояние между первой и N -й вершиной, то такая вершина не лежит ни на одном кратчайшем пути. Иначе число кратчайших путей равно произведению числа кратчайших путей от этой вершины до первой вершины и до N -й вершины. Эти числа легко считаются с помощью динамики по графу, можно даже это делать параллельно с подсчётом расстояний. Осталось только поделить число кратчайших путей, проходящих через вершину, на общее число кратчайших путей и умножить результат на два.

Задача H. Гранит науки

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

В университете, где учится Вася, начинается новый семестр. В новом семестре Васе предстоят занятия по N предметам, причём по каждому предмету занятия будут проходить каждый день. Перед началом семестра Вася узнал, сколько всего занятий планируется по каждому из предметов: по i -му предмету планируется N_i занятий для всех i от одного до N . Кроме того, от студентов старших курсов Вася узнал трудность каждого из занятий: число H_{ij} для всех i и j обозначает трудность j -го занятия по i -му предмету.

Многие преподаватели ещё не вернулись из отпусков, поэтому занятия по некоторым предметам начнутся не сразу. Если точнее, то первый день занятий по каждому из предметов выбирается равновероятно из первых M учебных дней. Может даже получиться так, что в первый учебный день вообще не будет занятий. После того как занятия по какому-то предмету начинаются, они проходят регулярно, по одному занятию в день, пока не будут проведены все N_i занятий.

Во время учёбы Вася устает. Вася посчитал, что за один день его усталость равна квадрату суммарной трудности всех занятий в этот день. Чтобы лучше понять, что ему предстоит, Вася хочет узнать, чему будет равна его суммарная усталость за весь семестр. Поскольку Вася ещё не знает, когда именно начнутся занятия, его интересует среднее значение.

Ограничения

$$1 \leq N \leq 500$$

$$1 \leq M \leq 500$$

$$1 \leq N_i \leq 500$$

$$0 \leq H_{ij} \leq 1000$$

Формат входного файла

Первая строка входного файла содержит два целых числа: N и M .

Каждая из следующих N строк соответствует одному предмету и содержит целое число N_i — число занятий по этому предмету, — и N_i целых чисел H_{ij} — трудность каждого из этих занятий.

Формат выходного файла

Выведите единственное число — среднюю суммарную усталость Васи за семестр. Выведите ответ с абсолютной или относительной погрешностью не более 10^{-7} .

Пример

stdin	stdout
2 1 1 1 1 2	9.0
3 3 1 3 2 1 1 3 0 0 0	14.3333333

Разбор задачи Н. Гранит науки

Пусть i -е занятие начинается в S_i -й день, а усталость Васи в i -й день равна $T_i = (\sum_j H_{j(i-S_j+1)})^2$, где суммирование происходит по всем j , для которых значение $H_{j(i-S_j+1)}$ определено. Нам необходимо найти $E(\sum_i T_i) = E(\sum_i (\sum_j H_{j(i-S_j+1)})^2) = \sum_i \sum_{j_1} E(H_{j_1(i-S_{j_1}+1)} \sum_{j_2} H_{j_2(i-S_{j_2}+1)})$. Если из суммы по j_2 вынести слагаемое, где $j_2 = j_1$, то оставшаяся часть будет независима от S_{j_1} , поэтому можно будет заменить среднее произведение произведением средних: $\sum_i \sum_{j_1} E(H_{j_1(i-S_{j_1}+1)} \sum_{j_2} H_{j_2(i-S_{j_2}+1)}) = \sum_i \sum_{j_1} (E(H_{j_1(i-S_{j_1}+1)}^2) + E(H_{j_1(i-S_{j_1}+1)}) \sum_{j_2 \neq j_1} E(H_{j_2(i-S_{j_2}+1)}))$.

Осталось научиться вычислять $E(H_{j(i-S_j+1)})$ и $E(H_{j(i-S_j+1)}^2)$ для всех i и j .

Они равны $\frac{\sum_{S_j=1}^M H_{j(i-S_j+1)}}{M}$ и $\frac{\sum_{S_j=1}^M H_{j(i-S_j+1)}^2}{M}$ соответственно. Используя скользящие суммы, можно вычислить эти значения за $O(\sum N_i)$. Чтобы быстро вычислять суммы по $j_2 \neq j_1$, можно из суммы по всем j вычесть значение для j_1 . Таким образом можно вычислить ответ за $O(MN + \sum N_i)$.

Задача I. Вероятный диагноз

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

В качестве курсовой работы Вася проектирует информационную систему больницы. Сейчас ему нужно написать компонент, который ставит предварительный диагноз на основе статистических данных.

В программу загружена информация об N болезнях и M возможных симптомах. Для каждой болезни известна вероятность того, что у больного будет проявляться каждый из симптомов. Аналогичная информация известна и для здоровых людей. Вася предполагает, что каждый из пациентов болен не более чем одной болезнью, а также то, что если зафиксировать болезнь (или её отсутствие), то различные симптомы будут проявляться независимо друг от друга.

В программу загружается информация о K пациентах. Для каждого из пациентов известно, что некоторые из симптомов у него проявляются, а некоторые нет, про некоторые же из симптомов ничего не известно. Нужно определить вероятность того, что он болен каждой из известных болезней.

Ограничения

$$1 \leq N \leq 200$$

$$\begin{aligned}1 &\leq M \leq 200 \\1 &\leq K \leq 200\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит два целых числа: N и M .

Следующая строка содержит M чисел — вероятность проявления каждого из симптомов у здорового человека.

Следующие N строк содержат информацию о болезнях. Каждая из них содержит $M + 1$ чисел: вероятность того, что человек болен этой болезнью (сумма этих вероятностей по всем болезням на превосходит 100 %), и вероятность проявления каждого из симптомов, если человек болен этой болезнью.

Следующая строка содержит единственное целое число K .

Следующие K строк содержат информацию о пациентах: каждая из них содержит строку длины M , каждый из символов которой равен +, если соответствующий симптом у пациента проявляется, -, если не проявляется, или ?, если о наличии этого симптома нет сведений.

Все вероятности во входном файле указаны в процентах, это числа от 0 до 100 ровно с двумя знаками после точки. Гарантируется, что комбинация симптомов для каждого пациента имеет ненулевую вероятность.

Формат выходного файла

Выполните K строк — по одной на каждого пациента. Каждая строка должна содержать N чисел — вероятности того, что соответствующий пациент болен каждой из болезней. Выполните вероятности как числа от 0 до 1 с абсолютной погрешностью не более 10^{-7} .

Пример

stdin	stdout
1 2	0.4000000
10.00 10.00	0.9818182
40.00 90.00 90.00	0.4000000
3	
??	
++	
+-	
2 3	0.0422797 0.0094707
10.00 30.00 50.00	0.4780115 0.0114723
10.00 50.00 50.00	0.0994036 0.0159046
50.00	0.0420408 0.0150674
1.00 20.00 30.00	
20.00	
4	
--?	
++?	
??-	

Разбор задачи I. Вероятный диагноз

Задача эта, в общем-то, на применение формулы Байеса. Нам известна априорная вероятность того, что пациент болен каждой из болезней, а условная вероятность того, что пациент имеет именно такие симптомы, получается путём перемножения вероятностей для отдельных симптомов (для тех симптомов, которые проявляются, используется вероятность того, что соответствующий симптом проявляется, для тех, которые не проявляются, соответственно, используется вероятность того, что симптом не проявляется). Задача состоит в том, чтобы найти условную вероятность того, что больной болен определённой болезнью по симптомам. Вероятности для каждого пациента вычисляются за $O(MN)$, таким образом общая сложность получается $O(MNK)$.

Задача J. Зоологический эксперимент

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

Вася занимается в зоологическом кружке и ставит там эксперименты над шушпанчиками. В одном из экспериментов он помещает двух шушпанчиков в лабиринт, который представляет собой неориентированный граф. Каждую секунду каждый из шушпанчиков выбирает одну из вершин лабиринта, смежную с текущей, и прыгает туда. Шушпанчики выбирают каждую из смежных вершин с равной вероятностью. Хотя они и находятся в одном лабиринте, они никак не реагируют друг на друга и движутся совершенно независимо. По крайней мере, Вася так считает. Чтобы проверить эту гипотезу, он решил измерить, какую часть времени, в среднем, шушпанчики проводят в одной и той же вершине. Чтобы избежать погрешности, Вася усредняет долю секунд, которую шушпанчики находились в одной вершине, за продолжительный период времени. Также Вася считает, что несмотря на то, что шушпанчики никак не реагируют друг на друга, они выдерживают ритм с такой точностью, что прыгали всё это время совершенно синхронно. Необходимо сделать теоретический расчёт этой величины.

Ограничения

$$\begin{aligned}2 &\leq N \leq 100 \\1 &\leq M \leq 10000\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит два целых числа: N и M — количество вершин и рёбер в лабиринте.

Каждая из следующих M строк содержит два целых числа — номера вершин (от 1 до N), соединённых ребром. Гарантируется, что в графе нет петель, параллельных рёбер и изолированных вершин.

Следующая строка содержит два целых числа от 1 до N — номера вершин, в которые были изначально помещены шушпанчики. Эти номера могут совпадать. Шушпанчики начинают движение одновременно.

Формат выходного файла

Выведите единственное число — долю времени, которую шушпанчики должны проводить в одной и той же вершине, если Васины предположения верны.

Пример

stdin	stdout
5 5	0.2
1 2	
2 3	
3 4	
4 5	
1 5	
1 2	

Разбор задачи J. Зоологический эксперимент

Эта задача призвана показать различные виды состояний в цепях Маркова. В начале следует проверить некоторые особые случаи. Во-первых, если шушпанчики начинают путь в различных компонентах связности, то ответ равен нулю. В дальнейшем будем рассматривать только ту компоненту, в которой находятся шушпанчики. Во-вторых, если эта компонента двудольная и шушпанчики находятся в различных долях, ответ также равен нулю.

Дальнейшее решение зависит от того, является ли компонента с шушпанчиками двудольной. Если нет, то несложно доказать, что соответствующая цепь Маркова является неразложимой и апериодической, а значит, у неё есть единственное стационарное состояние и система стремится к нему из любого начального состояния. Если же компонента является двусвязной, то цепь, хотя и остаётся неразложимой, будет периодической с периодом 2. В этом случае есть смысл разбить цепь на две: одну для чётных состояний, а другую для нечётных. Переходы внутри этих цепей будут делаться сразу на два шага. Дальше нужно и в том, и в другом случае найти стационарное состояние, например, решив систему уравнений методом Гаусса. Затем нужно просуммировать квадраты вероятностей и, если цепей две, взять среднее арифметическое (поскольку половину времени шушпанчики проводят в одной цепи, а половину в другой).

Задача K. Игра (Высшая лига)

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

Вася встретил приятеля, который предложил ему сыграть на коллекционные карты. Игра проходит в несколько раундов, и Вася рассчитал, что в

каждом раунде он может выиграть с вероятностью $P\%$. Если он выигрывает раунд, то приятель отдаёт ему N_1 коллекционных карт, иначе Вася отдаёт приятелю N_2 коллекционных карт. Когда проигравшему нечего платить, игра заканчивается, и тот, у кого закончились карты, считается проигравшим всю игру. У Васи M_1 коллекционных карт, и он знает, что у приятеля M_2 коллекционных карт. Чтобы решить, стоит ли играть, Вася хочет посчитать вероятность своего выигрыша при таких условиях, но ему это не под силу. Помогите ему.

Ограничения

$$\begin{aligned}1 &\leq N_1, N_2, M_1, M_2 \leq 50 \\0 &\leq P \leq 100\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит пять целых чисел: N_1, N_2, M_1, M_2 и P .

Формат выходного файла

Выведите единственное число — вероятность Васи выиграть всю игру. Выведите ответ с абсолютной погрешностью не более 10^{-7} .

Пример

stdin	stdout
1 1 10 10 50	0.5
2 1 10 10 40	0.8943674

Разбор задачи K. Игра

Хотя задача и кажется сложной, решается она достаточно просто — с помощью вероятностной динамики. В качестве состояния можно использовать, например, количество денег у Васи, тогда количество денег у приятеля определяется однозначно. Значение — вероятность выигрыша. Каждому состоянию соответствует линейное уравнение, которое соответствует тому, что после очередного хода средняя вероятность выигрыша не изменится. Систему уравнений можно решить методом Гаусса. Сложность решения $O((N_1 + N_2)^3)$.

Задача L. Опасная игра (Высшая лига)

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

Вася попал в серьёзную передрягу. Он задолжал мафии N долларов, и время расплаты почти подошло. Вася хорошо знает, чем грозит несвоевременный возврат долга, поэтому он хочет собрать N долларов любой ценой.

Однако, на данный момент у него есть только M долларов. Приятель, который узнал о проблемах Васи, предложил Васе игру — в этот раз на деньги. У Васи есть K попыток сделать ставку на любую сумму от нуля до той суммы, которая у него есть — не обязательно даже целую. С вероятностью $P\%$ Вася получает ставку в двойном объёме, иначе не получает ничего. Васе не важно, сколько денег у него останется в итоге, главное, чтобы было N долларов, чтобы расплатиться. Вася, конечно же, играет оптимально. Узнайте, с какой вероятностью ему повезёт и он расплатится деньгами.

Ограничения

$$0 \leq N \leq 10^9$$

$$0 \leq M \leq 10^9$$

$$1 \leq K \leq 12$$

$$0 \leq P \leq 100$$

Формат входного файла

Первая строка входного файла содержит четыре целых числа: N , M , K и P .

Формат выходного файла

Выведите единственное число — вероятность собрать достаточную сумму. Выведите ответ с абсолютной погрешностью не более 10^{-7} .

Пример

stdin	stdout
1000000 750000 1 10	0.1
1000000 750000 2 10	0.19

Разбор задачи L. Опасная игра

Сначала может показаться, что эту задачу невозможно решить, поскольку в ней бесконечно много состояний — ставки то могут быть и дробными.

Оказывается, что ответ задачи — кусочно-постоянная функция от M . Будем решать задачу с конца. Если $K = 0$, то если $M < N$, ответ равен нулю, иначе единице. Если $K = 1$, то отрезок делится на две части: если $0 \leq M < \frac{N}{2}$, то ответ равен нулю, если же $\frac{N}{2} \leq M < N$, то ответ равен P . Если же $M \geq N$, то ответ, как и раньше, равен единице.

Оказывается, эту закономерность можно обобщить. Для произвольного K отрезок от нуля до N делится на 2^K равных отрезков, и на каждом из них ответ постоянен. Доказывается это методом индукции по K : действительно, после каждой попытки можно считать, что начинается новая игра с K , меньшим на единицу. Отрезков, соответственно, будет в два раза меньше. Сделав ставку, Вася может перейти из одного отрезка в текущем состоянии в два отрезка в следующем — в один в случае выигрыша, а в другой в случае проигрыша. Пара отрезков зависит от текущего значения N и от ставки, сделанной Васей. Заметим, что из всех точек одного и того же отрезка в текущем состоянии достижимы одни и те же пары отрезков в следующем состоянии. Поскольку различные точки отрезков в следующем состоянии дают одинаковый выигрыш по предположению индукции, для отрезков в текущем состоянии это тоже будет верно.

Из доказательства получается и решение задачи: для каждого K и каждого отрезка будем перебирать все возможные переходы (их порядка 2^K). Так как состояний тоже порядка 2^K , общая сложность получается 2^{2^K} .

Задача М. Хеш-таблица (Высшая лига)

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

Недавно Вася узнал про новую структуру данных: хеш-таблицу с открытой адресацией. Хеш-таблица с открытой адресацией состоит из N ячеек, пронумерованных от 1 до N . Каждая из них может быть или свободна, или хранить какое-то значение. При вставке нового значения от него вычисляется хеш — случайное число от 1 до N (назовём его h) — после чего, если ячейка под номером h свободна, то значение записывается в неё. Иначе, если ячейка $h + 1$ свободна, то значение вставляется в неё, иначе проверяются ячейки $h + 2, h + 3$ и так далее. Если поиск дошёл до N -й ячейки, и она тоже занята, поиск продолжается с ячейки под номером 1. Таким образом, пока в таблице есть свободные места, значение так или иначе будет добавлено.

Каждая проверка ячейки, которая оказалась занята, называется коллизией. Например, если значение, хеш которого равен h , оказалось записанным в

ячейку $h+2$, то при его вставке произошли две коллизии. Коллизии замедляют работу хеш-таблицы, поэтому Вася решил узнать, сколько коллизий будет в среднем, если в пустую хеш-таблицу вставить M различных значений. А вычислять, как всегда, вам.

Ограничения

$$\begin{aligned}1 &\leq N \leq 100 \\0 &\leq M \leq N\end{aligned}$$

Формат входного файла

Первая строка входного файла содержит два целых числа: N и M .

Формат выходного файла

Выведите единственное число — среднее количество коллизий. Выведите ответ с абсолютной или относительной погрешностью не более 10^{-7} .

Пример

stdin	stdout
5 1	0.0
10 2	0.1

Разбор задачи М. Хеш-таблица

В условии задачи описана хеш-таблица, которая циклически замкнута. Рассмотрим разомкнутую хеш-таблицу. В ней, если произошла коллизия в последней ячейке, будем говорить, что произошло переполнение. Будем решать такую задачу: в разомкнутую хеш таблицу размера N вставили M элементов. Чему равна вероятность того, что переполнения не произойдёт, и сколько при этом в среднем будет коллизий?

Вначале рассмотрим случай, когда $M < N$. Чтобы найти вероятность, разделим одно событие “не произошло переполнение” на несколько отдельных событий, i -е из которых имеет вид “не произошло переполнения и в итоге первая свободная ячейка имеет индекс i ”. Эти события несовместимы, поэтому вероятность исходного события равна сумме вероятностей частей. Среднее количество коллизий также легко вычисляется как взвешенное среднее. Теперь заметим, что событие “не произошло переполнения и в итоге первая свободная ячейка имеет индекс i ” выполняется тогда и только тогда, когда в первые $i - 1$ ячеек попало ровно $i - 1$ хешей, в “подтаблице” из первых $i - 1$ ячеек не произошло переполнения, в i -ю ячейку не попало ни одного хеша, и в “подтаблице” из последних $N - i$ ячеек, в которую попало $M - i + 1$ хешей,

также не произошло переполнения. Таким образом, задача сводится к подзадачам с меньшим N . Вероятность того, что запросы лягут нужным образом, легко решается комбинаторными методами и равна $\frac{C_M^{i-1}(i-1)^{i-1}(N-i)^{M-i+1}}{N^M}$.

Если $N = M$, то перебирать нужно не первую свободную ячейку (таковых нет), а последнюю занятую. И опять задача делится на две с меньшими N . Чтобы свести исходную задачу (с замкнутой хеш-таблицей) к задаче с разомкнутой хеш-таблицей, нужно опять же, в зависимости от того, равны ли N и M или нет, перебрать или последнюю занятую ячейку, или первую и последнюю свободную ячейки.

Поскольку состояние динамики квадратичное, а количество переходов линейное, итоговая сложность $O(N^3)$.

День второй (16.02.2014 г.) Контест Сергея Нагина

Об авторе...

Нагин Сергей Юрьевич, родился 20 февраля 1995 года в городе Александрия. С 2001 по 2007гг. учился в школе №9 , а с 2007 по 2012 гг. — в лицее информационных технологий города Александрии. Сейчас учится в Киевском Национальном Университете имени Тараса Шевченко — факультет кибернетики, направление информатики. Интересы: музыка, спорт, активный отдых, путешествия, экстрим, пробовать что то новое и неизвестное, знакомства и общение с интересными людьми и многое многое другое.



Основные достижения:

- Участник IOI 2010.
- Серебряный медалист IOI 2011.
- Золотой медалист IOI 2012.
- В составе команды Unicon занял пятое место на SEERC 2012.
- В 2013 году участвовал в финале Russian CodeCup.
- Дал на Codeforces 10 контестов.

Теоретический материал. Поиск состояний в задачах на динамическое программирование

1. Сокращение массива

Допустим в задаче дан массив из n целых чисел a_1, a_2, \dots, a_n , нас просят найти некоторую величину, которая зависит от этого массива. Пускай выполнены два условия:

- не важен порядок следования чисел;

- числа не являются большими.

Тогда можно заменить массив на массив q , где q_i равно количеству чисел в массиве a , равных i . Такое преобразование с некоторой вероятностью может позволить написать некоторую динамику, которая работает с элементами массива q , как с параметрами.

1.1 Тривиальный вариант задачи

Дан массив, состоящий из n целых чисел, числа могут быть либо нулями, либо единицами. Происходит n ходов, на каждом из ходов требуется удалить одно из чисел, при этом нельзя удалять два раза подряд число 0. Посчитать количество способов удалить все числа.

Посчитаем q_0 — количество нулей в массиве и q_1 — количество единиц. Пускай $dp_{i,j,last}$ — количество способов убрать i нулей и j единиц при условии, что мы только что удалили число $last$. Такую динамику очень просто посчитать. Решением задачи будет $dp_{q_0,q_1,1}$.

1.2 Задача про расстановки чисел

Пускай у нас есть массив из n целых чисел. Нужно посчитать количество способов расставить эти числа, чтобы не было пары соседних одинаковых чисел ($n \leq 100$).

Для решения задачи заменим массив чисел на их количество вхождений, так как порядок их следования в данном массиве нам не интересен. Далее сделаем тоже самое еще раз, так как сами величины чисел нам не интересны, важно лишь отношение равенства. Теперь можно написать динамику на векторе (массиве, который получили). Как оказывается, состояний не будет много.

2. Задачи на оптимизацию линейной динамики

Пускай Вам дана задача, которую можно решить некоторой линейной динамикой. Но один из параметров в задаче является слишком большим (порядка 10^{18}). Рассмотрим как решать простейшую задачу из подобных.

2.1 Поиск n -го числа Фибоначчи

Как известно ряд Фибоначчи задаются следующим образом: $F_0 = 0$, $F_1 = 1$, $F_i = F_{i-1} + F_{i-2}$. Ясно, что для подсчета следующего элемента нам нужны только 2 предыдущих, остальные нас не интересуют. Запишем вектор из двух элементов (F_i, F_{i-1}) , вектор который мы хотим получить (F_{i+1}, F_i) . Подберем матрицу коэффициентов A , которая будет превращать первый вектор во второй:

- очевидно, что F_i должно переходить в F_{i+1} и F_i в количестве 1;
- F_{i-1} переходит в F_{i+1} в количестве 1;
- $a_{1,1} = 1, a_{1,2} = 1, a_{2,1} = 1, a_{2,2} = 0$;
- проверка: $F_{i+1} = 1 \cdot F_i + 1 \cdot F_{i-1}, F_i = 1 \cdot F_i$ — все верно, значит матрица переходов A подобрана верно.

Теперь имея начальный вектор (F_0, F_1) мы можем получить любой интересующий нас последовательным умножением стартового вектора на A . Такое решение все еще работает линейное время, но мы рассмотрим формулу для получения искомого вектора более детально: $V_n = (F_0, F_1) \cdot A \cdot A \cdots A = (F_0, F_1) \cdot A^{n-1}$. Так как для матриц выполняется свойство $(A \cdot B) \cdot C = A \cdot (B \cdot C)$, то мы можем посчитать A^{n-1} с помощью бинарного возведения в степень, а затем просто умножить вектор на результатирующую матрицу.

Нужно заметить, что решения с возведением матрицы в степень работают за время $O(\log(n) \cdot k^3)$, где n — количество итераций, а k — размерность матрицы.

2.2 Задача про двух солдат

Пускай на прямой находятся два солдата, изначально они стоят в начале координат. За одну единицу времени ровно один из них может пойти на один метр вперед, при этом должны выполняться следующие условия:

- первый солдат всегда стоит не ближе второго к началу координат;
- расстояние между первым и вторым солдатом не должно превышать 10 метров.

Посчитать количество последовательностей перемещения солдат, учитывая что после каждого их шага оба правила будут соблюдены, а после всех шагов они будут находиться в одной точке.

Медленное решение задачи, это динамика $dp_{i,dist}$, i — номер шага, $dist$ — расстояние между солдатами. Переходы? Очевидно, что $dp_{i,dist}$ переходит в $dp_{i+1,dist+1}$ и $dp_{i+1,dist-1}$ в количестве 1 (при учете, что новый $dist$ не выйдет за интервал $[0, 10]$).

Пускай d_i — вектор $(dp_{i,0}, dp_{i,1}, \dots, dp_{i,10})$, найдем матрицу коэффициентов A , которая будет приводить его к d_{i+1} . Для поиска матрицы просто запишем все переходы между элементами в ней. Повторим те же действия, что и при поиске n -го числа Фибоначчи, при условии, что стартовый вектор будет иметь вид $(1, 0, 0, \dots, 0)$, а матрицу нужно возводить в степень n (так как мы будем производить n ходов).

2.3 Объединим знания

Дан массив из n элементов, за один шаг можно увеличить ровно 1 элемент массива на 1 по модулю 3. Сколько существует способов сделать m ходов, что бы в результате в массиве содержалось q_0 нулей, q_1 единиц, q_2 двоек. ($n \leq 10$, $m \leq 10^{18}$, $0 \leq elements < 3$, $q_0 + q_1 + q_2 = n$).

Для решения заменим данный массив на массив w такой, что (w_0, w_1, w_2) описывает количество нулей, единиц и двоек в текущий момент. Заметим, что таких массивов не так и много в любой момент. Запишем все возможные состояния массива в вектор. Пускай A — матрица преобразований, при этом $A_{i,j}$ — количество способов перейти от вектора i к вектору j за один ход. Дальнейшее решения абсолютная копия предыдущих двух.

3. Замена параметра на результат

Для некоторых видов динамического программирования можно внести результат как параметр динамики, а параметр сделать результатом. Рассмотрим конкретно на примере задачи.

3.1 Максимальная возрастающая подпоследовательность

Есть массив b из t целых чисел. Массив a задается конкатенацией массива b t раз. Требуется найти максимальную возрастающую подпоследовательность в массиве a . Известно, что $t \cdot d(b) \leq 10^7$, где $d(b)$ — количество различных элементов в массиве b .

Не сложно придумать динамику dp_i — максимальную длину возрастающей подпоследовательности, заканчивающейся в элементе с номером i . Но мы внесем результат в параметр, а параметр вынесем в результат, получим: $dp_{i,j}$ — минимальная позиция, такая что возрастающая последовательность длины j заканчивается элементом, величина которого не превосходит i . Такую динамику уже не сложno пересчитать.

Задачи и разборы

Задача А. Сережа и уменьшения (Юниорская лига)

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

У Сережи есть массив из n целых чисел a_1, a_2, \dots, a_n . Сережа хочет уметь отвечать на два вида запросов:

- $0 \ l \ r \ e$ — для всех i ($l \leq i \leq r$) выполнить присваивание $a_i = a_i - e$
- $1 \ l \ r$ — узнать количество таких i ($l \leq i \leq r$), что $a_i \leq 0$

Помогите Сереже, ответьте на m запросов.

Формат входного файла

Первая строка содержит два целых числа n, m ($2 \leq n \leq 4000, 1 \leq m \leq 35000$). Следующая строка содержит n целых чисел a_1, a_2, \dots, a_n ($0 \leq a_i \leq 10^9$). Следующие m строк содержат запросы, в виде, как описано в условии. Если запрос имеет вид $0 \ l \ r \ e$, то выполняется ограничение $1 \leq l \leq r \leq n, 0 \leq e \leq 10^9$. Если запрос имеет вид $1 \ l \ r$, то выполняется ограничение $1 \leq l \leq r \leq n$.

Формат выходного файла

Для каждого запроса вида $1 \ l \ r$ выведите ответ в отдельной строке.

Пример

stdin	stdout
3 6	0
1 2 3	1
0 2 3 1	2
1 1 3	
0 2 2 1	
1 1 3	
0 1 3 1	
1 1 2	

Разбор задачи А. Сережа и уменьшения

См. решение аналогичной задачи высшей лиги.

Задача В. Сережа и шаблон (Юниорская лига)

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

У Сережи есть массив из n целых чисел a_1, a_2, \dots, a_n и стек.
Сережа умеет делать две операции:

- взять очередное число из массива a и добавить его в стек
- записать в конец нового массива верхний элемент стека и убрать его

Функция $f(a, b)$, где a и b — некоторые массивы целых чисел из n элементов равна количеству последовательностей операций длины $2 \cdot n$, которые из массива a строят массив b .

У Сережи есть массив из n целых чисел c_1, c_2, \dots, c_n . Некоторые элементы массива заменены на число -1 .

Ваша задача посчитать значение: $\sum_d f(a, d)$, где d — массив, который можно получить из c с помощью замен всех -1 на некоторые положительные числа.

Формат входного файла

Первая строка содержит целое число n ($1 \leq n \leq 16$). Следующая строка содержит n целых чисел a_1, a_2, \dots, a_n ($1 \leq a_i \leq 100$). Следующая строка содержит n целых чисел c_1, c_2, \dots, c_n ($1 \leq c_i \leq 100$ или $c_i = -1$).

Формат выходного файла

В единственную строку выведите ответ на задачу по модулю 1000000007 ($10^9 + 7$).

Пример

stdin	stdout
4 1 10 1 100 1 100 1 10	1
5 1 2 3 4 5 -1 -1 -1 -1 -1	42

Разбор задачи В. Сережа и шаблон

См. решение аналогичной задачи высшей лиги.

Задача С. Сережа и массив (Юниорская лига)

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

У Сережи есть массив из n целых чисел a_1, a_2, \dots, a_n . Сережа умеет делать следующую операцию:

- зафиксировать два целых числа l, r ($1 \leq l \leq r \leq n$);
- увеличить все a_i на число 1 ($l \leq i \leq r$).

Сережа ввел функцию $f(a, x)$ — минимальное число операций, чтобы все элементы массива a стали равны x .

Сейчас Сережу интересует количество массивов a длиной n , состоящих из целых неотрицательных чисел, не превосходящих m , таких, что $f(a, m) \leq k$.

Помогите Сереже, посчитайте искомое количество.

Формат входного файла

В единственной строке заданы целые числа n, m, k ($1 \leq n, m \leq 1000, 1 \leq k \leq 60$).

Формат выходного файла

В единственную строку выведите ответ на задачу по модулю 1000000007 ($10^9 + 7$).

Пример

stdin	stdout
5 3 4	672

Разбор задачи С. Сережа и массив

См. решение аналогичной задачи высшей лиги.

Задача D. Сережа и вода

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

У Сережи есть n резервуаров. Каждый резервуар имеет вместительность s литров. Так же у Сережи есть m бутылок воды с объемами a_1 литров, a_2 литров, \dots , a_m литров. Каждая бутылка полностью заполнена водой.

Сережа может открыть любую бутылку и вылить ее содержимое в некоторый из своих резервуаров. При этом, если резервуар уже полон в некоторый момент выливания воды, то остаток воды в бутылке выкидывается.

Сейчас Сережа хочет знать: сколько резервуаров можно полностью наполнить водой?

Помогите ему.

Формат входного файла

В первой строке заданы целые числа n , s и m ($1 \leq n, s, m \leq 1000$). Вторая строка содержит целые числа a_1, a_2, \dots, a_m ($1 \leq a_i \leq 3$).

Формат выходного файла

В единственную строку выведите ответ на задачу.

Пример

stdin	stdout
10 3 5 3 3 1 2 2	3
10 7 6 3 2 2 2 2 3	2

Разбор задачи D. Сережа и вода

Посчитаем количества бутылок каждого типа (q_1, q_2, q_3). Теперь напишем динамику от этих параметров. dp_{i_1, i_2, i_3} — пара чисел (x, y) , где x — максимальное число резервуаров, наполненных водой полностью, а y — максимальный остаток в одном из резервуаров. Пересчет динамики элементарен.

Сложность решения $O(n^3)$, при маленькой константе.

Задача Е. Сережа и игра

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

Сережа играет в игру. Игра состоит из уровней. Считается, что уровень пройден успешно, если игрок нажал b клавиш на клавиатуре. Всего на клавиатуре p клавиш. Для успешного прохождения можно нажать любые клавиши, главное — количество.

Если игрок успешно проходит уровень, то он получает столько очков, сколько он получил на предыдущем уровне + 1, при этом действует ограничение, что игрок не может получить более k очков за уровень (то есть, если игрок на предыдущем уровне получил k очков и на текущем успешно выполнил свое задание, то он все равно получит k очков). Если же игрок делает что-то не правильно (нажимает меньше или больше клавиш), то на текущем шаге он получает столько же очков, сколько на предыдущем уровне. Заметим, что во втором случае уровень так же считается пройденным (но не успешно). Для первого уровня будем считать, что на предыдущем игрок получил 0 очков. Игрок может в любой момент завершить игру.

Сейчас Сережу заинтересовал вопрос: сколько существует способов сыграть в игру, что в сумме он получит ровно s очков, и при этом обязательно успешно пройдет первый уровень.

Два способа считаются разными, если на каком-то уровне игрок нажал на различный набор клавиш. Так же два способа будут различными, если в них будут разное количество пройденных уровней.

Формат входного файла

В первой строке задано целые числа b, p, k, s ($1 \leq k, p, s \leq 10^5, 0 \leq b \leq p$).

Формат выходного файла

В единственную строку выведите остаток от деления искомого количества способов на число 1000000007 ($10^9 + 7$).

Пример

stdin	stdout
1 2 2 3	12

Пояснение

В примере существует 12 способов получить сумму 3. Вот некоторые из

этих вариантов: ((1),(1)), ((2),(1)), ((2),(2)), ((1),(1,2),()), ((2),(),()). В скобках указаны номера кнопок нажатых на очередном уровне.

Разбор задачи E. Сережа и игра

Подсчитаем q — количество правильных нажатий и w — количество неправильных нажатий. Эти числа — биномиальные коэффициенты. Дальше будем пользоваться динамикой $dp_{i,j}$ — сколько способов набрать сумму i при том, что последний раз мы брали j очков. Зная q, w динамика пересчитывается очень просто.

Так как $j \cdot (j + 1) \leq 2 \cdot n$ сложность решения выходит $O(n^{3/2})$.

Задача F. Сережа и столбики

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

У Сережи есть n выставленных в ряд столбиков с высотами a_1, a_2, \dots, a_n . Сережа хочет k раз увеличить высоту некоторого столбика на 1, так чтобы величина $\left| \sum_{i=1}^{n-1} a_{i+1} - a_i \right|$ была минимальна.

Помогите Сереже, найдите эту минимальную величину.

Формат входного файла

Первая строка содержит целое число n и k ($1 \leq n \leq 10^5, 0 \leq k \leq 10^9$). Вторая строка содержит n целых чисел a_1, a_2, \dots, a_n ($0 \leq a_i \leq 10^9$).

Формат выходного файла

В единственную строку выведите искомую величину.

Пример

stdin	stdout
6 2	3
3 4 5 6 7 8	

Разбор задачи F. Сережа и столбики

Следует заметить, что ответ зависит только от числа a_1 и a_n . Следует отдельно рассмотреть три случая:

- $n = 1$
- $n = 2$
- $n > 2$

Задача G. Сережа и деление

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

Сережа считает, что массив из n целых чисел a_1, a_2, \dots, a_n сопоставим с числом c , если $(a_1 \cdot a_2 \cdots \cdot a_n) \bmod c = 0$.

Пускай $f(a, m)$ — количество таких x ($1 \leq x \leq m$), что массив a сопоставим с числом x .

Сейчас Сережу интересует сумма величин $f(a, m)$, для всех массивов таких, что их длина n , а все элементы целые и $1 \leq a_i \leq m$ ($1 \leq i \leq n$).

Формат входного файла

Единственная строка содержит два целых числа n и m ($1 \leq n \leq 10^6, 1 \leq m \leq 1000$).

Формат выходного файла

В единственную строку выведите остаток от деления ответа на число $1000000007 (10^9 + 7)$.

Пример

stdin	stdout
5 3	665

Разбор задачи G. Сережа и деление

Найдем все делители t . Запишем простую динамику $dp_{i,j}$ — количество способов построить массив из i элементов, что $\gcd(t, a_1 \cdot a_2 \cdots \cdot a_i) = j$. Построим матрицу переходов между делителями за один ход. Дальше воспользуемся методом, рассказанным на лекции.

Задача Н. Сережа и последовательность

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	4 Мб

У Сережи есть набор любимых чисел. Как-то раз, он взял листик и записал каждое любимое число ровно k раз. Подлый Дима выбрал одно из любимых чисел Сережи и стер несколько (как минимум одно, но не все) его вхождений с бумажки.

Сережа просит найти стертное число.

Помогите Сереже.

Будьте осторожны, в этой задаче очень ограничен лимит памяти.

Формат входного файла

В первой строке заданы целые числа n и k ($1 \leq k \leq n \leq 3 \cdot 10^6$). Вторая строка содержит целые числа a_1, a_2, \dots, a_n — числа записанные на бумажке ($1 \leq a_i \leq 10^9$).

Формат выходного файла

В единственную строку выведите ответ на задачу.

stdin	stdout
42 18 318752492 630995896 557490717 182588146 174325727 95851943 318752492 154469437 367325359 821370300 174325727 154469437 59827913 36526605 9356383 545129895 182588146 557490717 545129895 367325359 202714501 613423725 545129895 613423725 630995896 862808465 124864154 745008828 202714501 745008828 124864154 9356383 545129895 545129895 545129895 36526605 95851943 821370300 472683539 59827913 472683539 862808465	528363263

Разбор задачи Н. Сережа и последовательность

Для каждого двоичного бита посчитаем, в скольких числах он встречается, если остаток от деления этого количества на k не равен нулю, то он присутствует в ответе.

Задача I. Сережа и экзамен

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

У Сережи экзамен. Всего на экзамен выносится n вопросов, каждый вопрос принадлежит одной из t тем. В каждой теме есть хотя бы один вопрос.

В билете на экзамене будет присутствовать m вопросов, по одному с каждой темы. В каждой теме вопрос выбирается случайным образом.

Перед экзаменом Сережа не знает ни одного вопроса. За ночь Сережа может выучить k вопросов.

Теперь Сереже интересно: сколько существует способов выучить k вопросов, чтобы математическое ожидание его оценки было максимально. Оценка Сережи — это количество правильно отвеченных вопросов в билете.

Формат входного файла

Первая строка содержит целые числа n, m, k ($1 \leq n, m, k \leq 1000$). Вторая строка содержит целые числа a_1, a_2, \dots, a_n . a_i — номер темы вопроса номер i ($1 \leq a_i \leq m$).

Формат выходного файла

В единственную строку выведите ответ на задачу по модулю 1000000007 ($10^9 + 7$).

Пример

stdin	stdout
10 3 4 1 1 1 2 2 3 3 3 3 3	3

Разбор задачи I. Сережа и экзамен

Выгоднее всего учить билеты, чья тема содержит минимальное количество вопросов. Если у нас неоднозначен выбор между некоторыми билетами, то можно выбрать любой. Для подсчета ответа нужно найти те неоднозначные билеты и посчитать количество способов выбора нужного количества билетов из тех, что имеем, это ни что иное, как число сочетаний.

Задача J. Сережа и уменьшения (Высшая лига)

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

У Сережи есть массив из n целых чисел a_1, a_2, \dots, a_n . Сережа хочет уметь отвечать на два вида запросов:

- $0 \ l \ r \ e$ — для всех i ($l \leq i \leq r$) выполнить присваивание $a_i = a_i - e$;
- $1 \ l \ r$ — узнать количество таких i ($l \leq i \leq r$), что $a_i \leq 0$.

Помогите Сереже, ответьте на m запросов.

Формат входного файла

Первая строка содержит два целых числа n, m ($2 \leq n \leq 10^5, 1 \leq m \leq 2 \cdot 10^5$). Следующая строка содержит n целых чисел a_1, a_2, \dots, a_n ($0 \leq a_i \leq 10^9$). Следующие m строк содержат запросы, в виде как описано в условии. Если запрос имеет вид $0 \ l \ r \ e$, то выполняется ограничение $1 \leq l \leq r \leq n$, $0 \leq e \leq 10^9$. Если запрос имеет вид $1 \ l \ r$, то выполняется ограничение $1 \leq l \leq r \leq n$.

Формат выходного файла

Для каждого запроса вида $1 \ l \ r$ выведите ответ в отдельной строке.

Пример

stdin	stdout
3 6	0
1 2 3	1
0 2 3 1	2
1 1 3	
0 2 2 1	
1 1 3	
0 1 3 1	
1 1 2	

Разбор задачи J. Сережа и уменьшения

Заведем дерево отрезков, которое сможет делать следующие виды операций:

- отнять на отрезке;
- найти минимум на отрезке.

Когда мы будем обрабатывать первый вид операции, то нужно выполнить следующую последовательность операций:

- отнять на отрезке данное число;

- найти минимум на этом отрезке и если он меньше нуля, то сначала пометить этот элемент как плохой, а потом заменить его на 10^{18} .

Когда нам приходит запрос на количество, то просто выведем количество помеченных элементов на отрезке. Это можно делать с помощью любой структуры данных.

Так как каждый элемент не будет помечен больше одного раза, итоговая сложность выходит $O((n + m) \cdot \log(n))$.

Задача K. Сережа и шаблон (Высшая лига)

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

У Сережи есть массив из n целых чисел a_1, a_2, \dots, a_n и стек.

Сережа умеет делать две операции:

- взять очередное число из массива a и добавить его в стек;
- записать в конец нового массива верхний элемент стека и убрать его.

Функция $f(a, b)$, где a и b — некоторые массивы целых чисел из n элементов равна количеству последовательностей операций длины $2 \cdot n$, которые из массива a строят массив b .

У Сережи есть массив из n целых чисел c_1, c_2, \dots, c_n . Некоторые элементы массива заменены на число -1 .

Ваша задача посчитать значение: $\sum_d f(a, d)$, где d — массив, который можно получить из c с помощью замен всех -1 на некоторые положительные числа.

Формат входного файла

Первая строка содержит целое число n ($1 \leq n \leq 100$). Следующая строка содержит n целых чисел a_1, a_2, \dots, a_n ($1 \leq a_i \leq 100$). Следующая строка содержит n целых чисел c_1, c_2, \dots, c_n ($1 \leq c_i \leq 100$ или $c_i = -1$).

Формат выходного файла

В единственную строку выведите ответ на задачу по модулю 1000000007 ($10^9 + 7$).

Пример

stdin	stdout
4 1 10 1 100 1 100 1 10	1
5 1 2 3 4 5 -1 -1 -1 -1 -1	42

Разбор задачи К. Сережа и шаблон

Переформулируем задачу: нужно посчитать количество последовательностей операций со стеком, чтобы результат удовлетворял шаблон. Такую задачу можно решить следующей динамикой: $dp_{i,j,len}$ — ответ на исходную задачу, когда $a = a[i \cdots i + len - 1]$, $b = b[j \cdots j + len - 1]$. Для пересчета будем перебирать: к какому символу шаблона мы хотим приравнять первый символ последовательности. Дальше задача распадается на две независимых.

Сложность решения $O(n^4)$.

Задача L. Сережа и массив (Высшая лига)

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

У Сережи есть массив из n целых чисел a_1, a_2, \dots, a_n . Сережа умеет делать следующую операцию:

- зафиксировать два целых числа l, r ($1 \leq l \leq r \leq n$);
- увеличить все a_i на число 1 ($l \leq i \leq r$).

Сережа ввел функцию $f(a, x)$ — минимальное число операций, чтобы все элементы массива a стали равны x .

Сейчас Сережу интересует количество массивов a длиной n , состоящих из целых неотрицательных чисел, не превосходящих m , таких, что $f(a, m) \leq k$.

Помогите Сереже, посчитайте искомое количество.

Формат входного файла

В единственной строке заданы целые числа n, m, k ($1 \leq n, m \leq 10^4, 1 \leq k \leq 100$).

Формат выходного файла

В единственную строку выведите ответ на задачу по модулю 1000000007 ($10^9 + 7$).

Пример

stdin	stdout
5 3 4	672

Разбор задачи L. Сережа и массив

Пускай у нас есть массив из целых чисел a_1, a_2, \dots, a_n . Тогда для его приведения нам нужно $\sum_{i=1}^{n-1} (a_{i+1} - a_i + |a_{i+1} - a_i|) + m - a_n$ операций. Теперь напишем динамику $dp_{i,j,l}$ — сколько есть массивов из i элементов с последним элементом j и текущей суммой операций l . Заметим, что если $m > k$, то можно сделать $m = k$.

С помощью частичных сумм доведем сложность решения до $O(n \cdot \min(m, k) \cdot k)$.

День третий (17.02.2014 г.) Конкурс Алексея Шмелева

Об авторе...

Алексей Шмелев, родился 23 августа 1988 года. Школа: МОУ Лицей #36 города Нижнего Новгорода (1995–2005 гг). ВУЗ: ННГУ им Н.И.Лобачевского, факультет вычислительный математики и кибернетики, специальность: прикладная информатика (2005–2009), магистратура (2009–2011).

Основные достижения:

- Победитель турнира ICL 2011 (в составе команды ННГУ).
- Серебряный призер финала ACM ICPC 2011 (в составе команды ННГУ).
- Участник онсайта Google Code Jam 2011.
- Участник онсайта Challenge-24 2012 (в составе команды _NiN_).
- Участник финала ACM ICPC 2012 (в составе команды ННГУ).
- Участник онсайта КРОК 2013.
- Участник онсайта Russian Code Cup 2013.
- Участник онсайта Challenge-24 2013 (в составе команды _NiN_).
- Рейтинг Codeforces: 2424.
- Рейтинг TopCoder: 2414.



Теоретический материал. Лемма Бёрнсайда и Теорема Пойа

Лемма Бёрнсайда

Опубликована в 1897 году Уильямом Бёрнсайдом.

Пусть у нас есть некоторые объекты (например, раскраски вершин квадрата или множество перестановок из N элементов), и мы можем выполнять некоторые операции над ними (например, поворачивать квадрат или циклически сдвигать перестановку). Объекты, получающиеся при помощи данных операций, будем считать эквивалентными исходному объекту. Лемма Бёрнсайда предоставляет способ нахождения количества “различных” объектов множества при определенном наборе действий.

Для того, чтобы воспользоваться математическими свойствами объектов, необходимо формализовать использующиеся понятия. Будем считать, что наши объекты образуют множество элементов — M . Далее будем считать, что каждое действие переводит элемент множества M в объект, тоже являющийся элементом M . Таким образом, получаем определение.

Определение 1: преобразованием множества M будем называть отображение, при котором каждому элементу $x \in M$ соответствует ровно один элемент этого же множества — образ x . Причем каждый элемент y является образом ровно одного элемента из M .

Если имеется несколько различных преобразований, то для каждого элемента $x \in M$ мы получаем столько же образов y , которые получаются при действии на x преобразованиями множества. Получаем второе определение.

Определение 2: если есть множество преобразований G на множестве M , то орбитой элемента x относительно множества преобразований называется множество образов x при всех преобразованиях из G :

$$O(x) = \{g(x) | g \in G\}$$

Например, для множества перестановок из N элементов можно задать множество преобразований — циклические сдвиги на $0, 1, \dots, N - 1$. При этом для каждой перестановки орбитой будет N перестановок, получающихся из данной при помощи циклического сдвига.

Далее хотелось бы наложить некоторые условия на множество преобразований так, чтобы получившиеся орбиты не пересекались между собой. Таким образом, множество преобразований G должно задавать отношение эквивалентности на множестве элементов M . Для этого достаточно соблюдения трех свойств:

1. Наличие тождественного преобразования во множестве G .
2. Наличие в G обратного преобразования для каждого преобразования из G .

3. Для каждой пары f и g преобразований из G , G так же содержит композицию преобразований $g(f)$.

Определение 3: множество преобразований, обладающее указанными тремя свойствами будем называть группой преобразований.

Далее будем работать только с группами преобразований. Если нам дано некоторое множество элементов M и группа преобразований G на этом множестве, то логично поставить вопрос, на сколько же классов эквивалентности разбивает группа G множество M ? Оказывается, для нахождения значения количества орбит $O(M, G)$ достаточно знать для каждого преобразования из G множество неподвижных элементов из M при данном преобразовании. А именно, верна следующая формула (лемма Бёрнсайда):

$$O(M, G) = \frac{\sum_{f \in G} x(f)}{|G|}$$

Где, $x(f)$ — количество неподвижных элементов множества M при преобразовании f , а $|G|$ — количество преобразований в группе G .

Пример 1: найти количество различных перестановок из N элементов при циклических сдвигах. Всего перестановок $N!$. Ни одно из преобразований, кроме тождественного (циклического сдвига на 0), не имеет неподвижных элементов. Тождественное преобразование имеет $N!$ неподвижных элементов. Всего преобразований — N . Таким образом, получаем очевидный результат:

$$O = \frac{N!}{N} = (N - 1)!$$

Пример 2: найти количество геометрически различных раскрасок вершин куба в 2 цвета (черный и белый) таких, что каждым цветом раскрашено ровно 4 вершины.

Всего возможных раскрасок вершин куба в 4 белые и 4 черные — $C_8^4 = 70$. Рассмотрим возможные движения и количества неподвижных элементов в них.

- Тождественное преобразование — 70 элементов.
- Поворот на ± 90 градусов вокруг оси, проходящей через центры противоположных граней (см. Рисунок 1). Подобных поворотов $3 \cdot 2$ (три пары противоположных граней, два направления поворота). При этом вершины грани $ABCD$ сдвинутся циклически, значит, они должны быть одного цвета. Аналогично, одного цвета должны быть вершины A_1, B_1, C_1, D_1 . Таким образом, есть два варианта раскраски куба:

1. грань $ABCD$ белого цвета, грань $A_1B_1C_1D_1$ — черного.
2. грань $ABCD$ черного цвета, грань $A_1B_1C_1D_1$ — белого.

Итого, получаем 6 преобразований и $6 \cdot 2 = 12$ элементов.

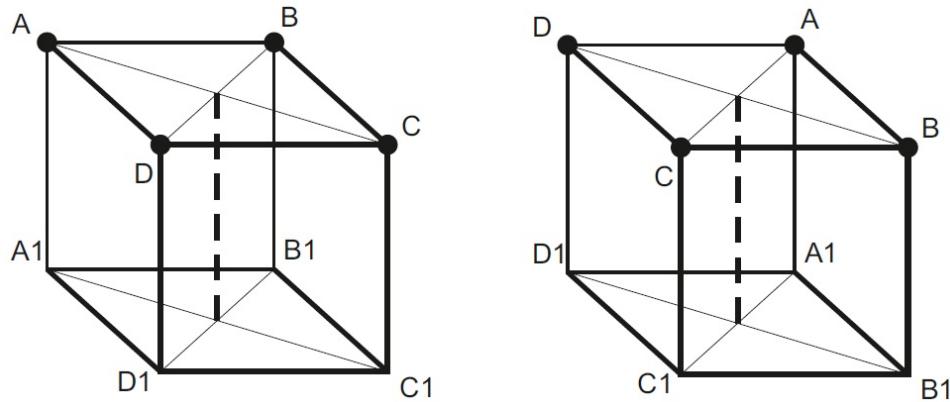


Рисунок 1

- Поворот на 180 градусов вокруг оси, проходящей через центры противоположных граней (см. Рисунок 2). Подобных поворотов 3. В каждом из них диагонали граней переходят сами в себя, но вершины каждой диагонали меняются местами. Значит, вершины диагонали должны быть одного цвета. Итого, у нас есть 4 диагонали, 2 из которых должны быть белыми. Значит, неподвижных вариантов раскраски будет $C_4^2 = 6$. Итого, получаем 3 преобразования и $6 \cdot 3 = 18$ неподвижных элементов.

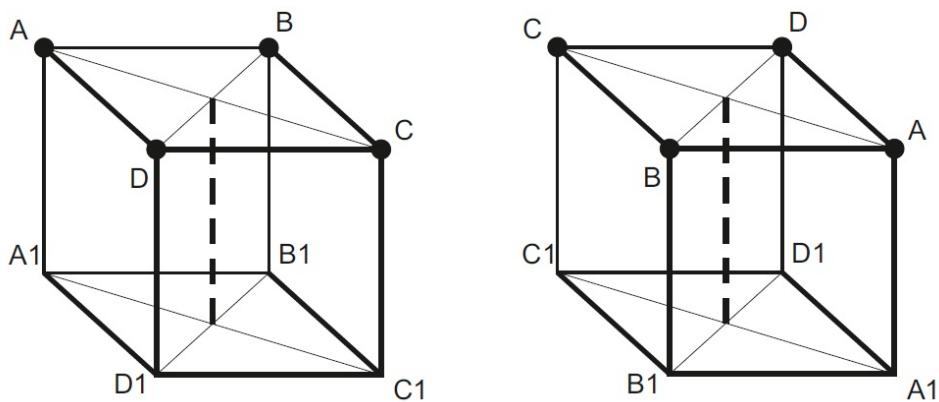


Рисунок 2

- Поворот на 180 градусов вокруг оси, проходящей через центры противоположных ребер (см. Рисунок 3). Подобных поворотов 6. При этом вершины грани $ABCD$ переходят в вершины грани $A_1B_1C_1D_1$. Для каждой раскраски грани $ABCD$ существует ровно одна раскраска грани $A_1B_1C_1D_1$, при которой раскраска куба перейдет сама в себя. При этом $A_1B_1C_1D_1$ будет содержать столько же черных вершин, сколько и $ABCD$. Значит, грань $ABCD$ должна содержать ровно 2 черных вершины, и возможных раскрасок $C_4^2 = 6$. Итого, получаем 6 преобразований и $6 \cdot 6 = 36$ элементов.

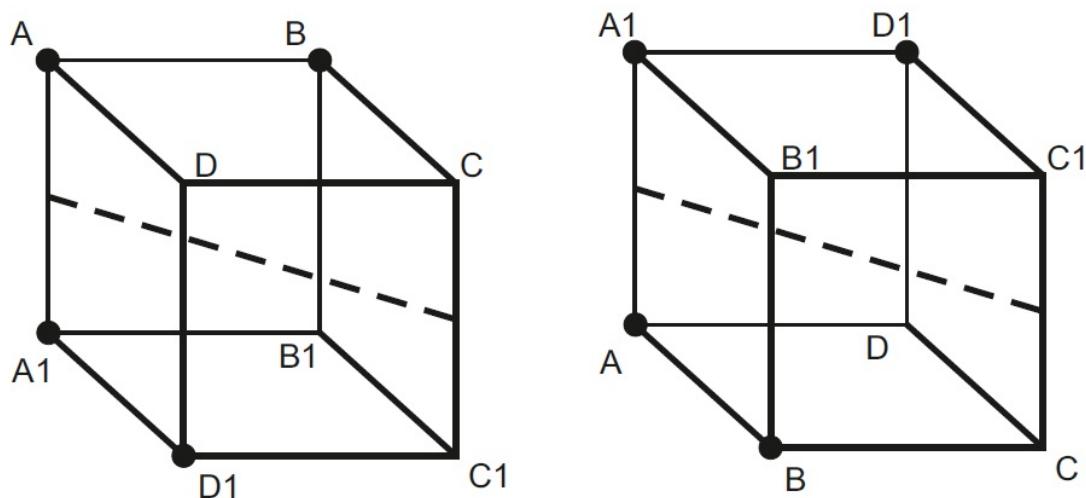


Рисунок 3

- Поворот на ± 120 градусов вокруг диагонали (см. Рисунок 4). Подобных поворотов 8. Вершины A, B_1, D_1 циклически сдвигаются, так же как и вершины B, C_1, D . Вершины A_1 и C остаются на месте. Значит, вершины A, B_1, D_1 должны быть одного цвета, вершины B, C_1, D — другого цвета. Тогда вариантов раскраски $2 \cdot 2$ (2 варианта цвета вершин A, B_1, D_1 и 2 варианта цвета вершины A_1). Итого, получаем 8 (4 диагонали и 2 угла поворота) преобразований и $8 \cdot 4 = 32$ неподвижных элемента.

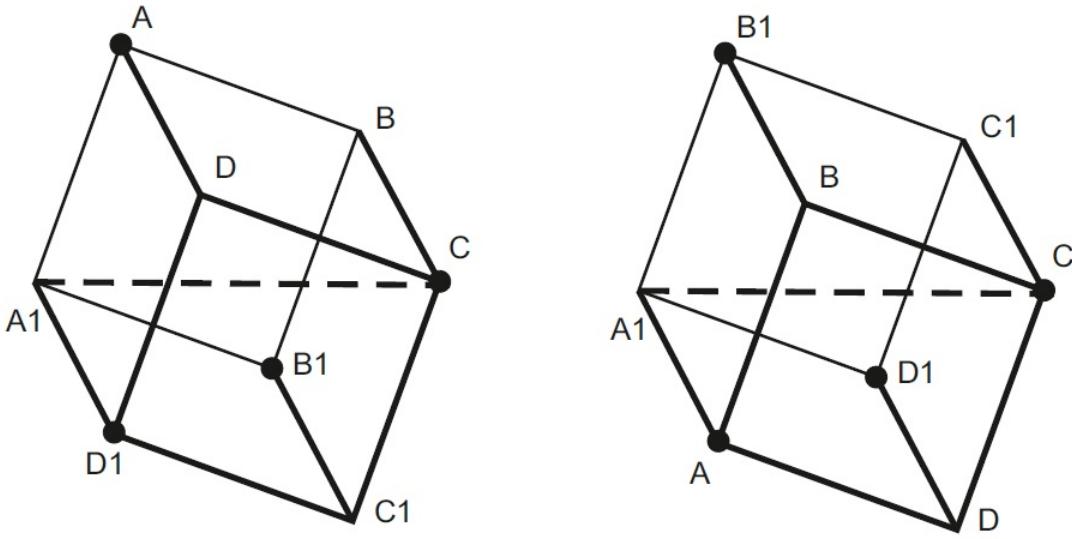


Рисунок 4

Всего мы получили 24 преобразования и 168 неподвижных элементов. Таким образом, число орбит равно $168/24 = 7$.

Пример 3: найти количество геометрически различных раскрасок тора в 2 цвета (источник задачи и реализация на языке C++ — http://e-maxx.ru/algo/burnside_polya). Множество M элементов — множество раскрасок доски $n \times m$ в два цвета. При этом множество преобразований будет содержать все композиции циклического сдвига вдоль строк, циклического сдвига вдоль столбцов и поворота доски на 180 градусов. При этом каждая композиция представляет собой некоторую перестановку клеток доски P , получающуюся при перемножении соответствующих “элементарных” перестановок композиции. Легко проверить, что количество неподвижных элементов при действии перестановки $P - 2^{C(P)}$, где $C(P)$ — количество циклов в перестановке. Посчитав количество различных перестановок и для каждой из них найдя количество циклов, можно найти количество различных раскрасок (с точностью до циклического сдвига и переворота) по лемме Бёрнсайда.

Пример 4: найти количество пар сопряженных перестановок.

Можно применить формулу по-другому: найти количество неподвижных элементов, зная число орбит множества M при действии группы преобразований G .

Пусть нам требуется найти количество пар перестановок A и B из N элементов, таких что выполняется: $A \cdot B = B \cdot A$. Это равенство можно записать

в виде $A = B \cdot A \cdot B^{-1}(\ast)$.

Тогда будем считать, что множество M состоит из всех перестановок A из N элементов. Множество преобразований $f \in G$ — действие перестановки B на перестановку A : $f(A) = B \cdot A \cdot B^{-1}$. Тогда искомое количество пар сопряженных перестановок — это суммарное количество неподвижных элементов для всех преобразований. Для его вычисления необходимо знать общее число преобразований и количество орбит, на которые разбиваются перестановки при таких преобразованиях. Всего преобразований — $N!$ (любая перестановка дает преобразование). При действии сопряженной перестановки цикловая структура перестановки не меняется. Общее число орбит — количество различных цикловых структур для перестановок из N элементов. Количество цикловых структур — это количество способов разбить число N на слагаемые — $S(N)$. Таким образом, общее число пар сопряженных перестановок равно $S(N) \cdot N!$

Задачи и разборы

Задача A. Who Calls the Crystal Maiden? (Юниорская лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася тренируется играть в Dota2. Сейчас он играет за героя Crystal Maiden — юную жрицу холода. Самым сильным заклинанием героя является Freezing Field. Во время действия заклинания вокруг Crystal Maiden происходят ледяные взрывы. Каждый взрыв наносит урон всем врагам на расстоянии не более r от центра взрыва. Центр каждого взрыва находится на расстоянии не более R от Crystal Maiden, причем местоположение центра взрыва равновероятно распределено в соответствующем круге. Всего происходит N взрывов. Взрывы независимы между собой, то есть положение одного взрыва никак не скаживается на положении следующих взрывов. Вася встретил вражеского героя и знает, что для убийства этого героя необходимо, чтобы его затронули хотя бы K взрывов заклинания Freezing Field. Зная положение вражеского героя, определите вероятность того, что он будет убит Васей.

Формат входного файла

Первая строка содержит два целых числа: $1 \leq r, R \leq 1000$ — радиус взрыва и радиус действия заклинания соответственно. Во второй строке находятся два целых числа $1 \leq N, K \leq 1000$ — общее число взрывов при применении

заклинания и число взрывов, необходимое для убийства вражеского героя. В последней строке находятся два целых числа $-1000 \leq X, Y \leq 1000$ — координаты вражеского героя относительно Crystal Maiden.

Формат выходного файла

Необходимо вывести единственное вещественное число с абсолютной погрешностью не более 10^{-6} — вероятность того, что вражеский герой будет убит в результате применения заклинания.

Пример

stdin	stdout
2 10 1 1 2 3	0.0400000000
2 10 2 1 2 3	0.0784000000

Разбор задачи A. Who Calls the Crystal Maiden? (Юниорская лига)

Взрыв заденет героя, если центр взрыва будет находиться на расстоянии не более r от положения героя. Значит, множество точек, попадание центра взрыва в которые поражает героя — это круг радиуса r с центром в точке (x, y) — положении героя. Вероятность того, что центр взрыва попадет в этот круг — отношение площади пересечения данного круга с областью действия заклинания к площади всей области заклинания. Таким образом, нужно найти площадь S пересечения кругов (x, y, r) и $(0, 0, R)$ и вероятность P поражения героя взрывом будет $P = S/(Pi * R * R)$. Теперь осталось найти вероятность того, что герой будет задет хотя бы K взрывами из N . Для этого можно использовать динамическое программирование, в котором состоянием $d[i, j]$ будет вероятность того, что из i событий выполняться хотя бы j при вероятности выполнения события P . Очевидно, что $d[i, j] = d[i - 1, j - 1] \times P + d[i - 1, j] * (1 - P)$. Найденное значение $d[N, K]$ и будет ответом на задачу.

Задача В. World of Dota: Cross (Юниорская лига)

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

Вася играет в Dota 2, но ему надоело воевать каждый раз на одной и той же карте. Поэтому Вася решил сделать новую карту для этой игры. Для простоты будем полагать, что карта представляет собой прямоугольник 9×9 , каждая клетка которого может быть либо свободна, либо занята деревом. Вася хочет, чтобы вражеская команда (состоящая из пяти героев) не могла группироваться близко к одной точке. Поэтому на карте не должно быть пяти свободных клеток, образующих крест с шириной и высотой 3:

```
. * .  
***  
. * .
```

С другой стороны, Вася хочет оставить как можно больше пространства для маневров, поэтому деревьев должно быть как можно меньше. Помогите Васе найти идеальную карту.

Формат входного файла

Эта задача не содержит входных данных.

Формат выходного файла

Необходимо вывести 9 строк, по 9 символов в каждой строке — искомую карту для Васи. Свободная клетка должна обозначаться символом '.', занятая деревом — '#'. Ответ считается верным, если карта является допустимой (не содержит крестов, состоящих из пяти свободных клеток) и содержит наименьшее число деревьев среди всех возможных допустимых карт.

Пояснение

Пример оптимальной допустимой карты 4×4 :

```
....  
. # ..  
. . #.  
....
```

Разбор задачи B. World of Dota: Cross (Юниорская лига)

Поскольку в задаче нужно вывести лишь одну конструкцию, то можно было запустить какой-либо перебор для поиска этой конструкции, либо составить ее «вручную». Оптимальная конструкция содержит 12 деревьев. Пример:

```
.....
...#.#...
.#.....#.
....#....
..#...#..
....#....
.#.....#.
...#.#...
.....
```

Задача C. Warlock (Юниорская лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася тренируется играть в Dota2. Сейчас он играет за героя Warlock. Одной из способностей героя является Fatal Bounds — герой «связывает» несколько вражеских юнитов вместе, заставляя их получать дополнительный урон. Заклинание работает следующим образом: если один из связанных юнитов получает урон (основной урон), то все остальные юниты получают P процентов от полученного юнитом урона (дополнительный урон). Действие связи не распространяется на дополнительный урон, полученный в результате действия заклинания (дополнительный урон не вызывает получение дополнительного урона остальными юнитами). Юнит, получивший основной урон, не получает дополнительный урон. Если юнит погибает при получении основного урона (количество очков жизни становится неположительным), то дополнительный урон будет рассчитываться от количества единиц жизни юнита перед получением основного урона, а не от изначального значения получаемого урона.

Вася хочет убить вражеского героя, поэтому связал его с N крипами при помощи Fatal Bounds. Далее будем считать, что вражеский герой не получает основной урон, в то время как все крипы будут воевать с Васей, пока не будут убиты. При этом часть полученного ими урона будет переходить Васе как дополнительный урон. Разумеется, сами крипы тоже будут страдать от по-

лучения дополнительного урона от Fatal Bounds. Вася может атаковать крипов произвольное число раз, при этом каждый раз он выбирает одного живого крипа, которому будет наноситься урон, и величину урона (положительное число). Вася атакует крипов, пока они не будут убиты. Зная изначальное количество очков здоровья у крипов, определите, какой наибольший урон может при этом получить вражеский герой.

Формат входного файла

В первой строке входного файла содержатся два целых числа: N — количество крипов, связанных с вражеским героем, $1 \leq N \leq 100000$ и P — процент дополнительного урона относительного основного, $0 \leq P \leq 100$. Во второй строке содержатся N натуральных чисел, не превосходящих 1000 — количество единиц здоровья у крипов, связанных Fatal Bounds.

Формат выходного файла

Необходимо вывести одно вещественное число с абсолютной или относительной погрешностью не более 10^{-6} — наибольший урон, который может получить вражеский герой.

Пример

stdin	stdout
2 20	21.600000000
100 10	

Разбор задачи C. Warlock (Юниорская лига)

Если в некоторый момент времени один из крипов получит D урона, при этом осталось M живых крипов, то герой получит $P*D$ единиц урона, а крипы в сумме получат $D + (M - 1)*P*D$ единиц урона. Понятно, что для получения героя максимального урона необходимо, чтобы в каждый момент нанесения урона живых крипов было как можно меньше, в этом случае здоровье крипов будет «расходоваться» наиболее эффективно. Значит, нужно убить первого крипа, нанеся как можно меньше урона. Стало быть, сначала надо атаковать крипа с наименьшим количеством здоровья, пока он не умрет. Аналогично, затем нужно будет убить крипа с наименьшим здоровьем из оставшихся и так далее. Значит, сначала нужно упорядочить крипов по количеству начального здоровья и убивать их в порядке возрастания. Поддерживая суммарный нанесенный урон D , на каждом шаге нужно наносить еще $H[i] - D$ урона, где $H[i]$ — здоровье очередного крипа.

Задача D. Wild Card: Subway

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася долго учился играть в Dota2, но его команда так и не смогла пробиться на финал Starladder 2014. Вася был жутко расстроен и подумывал о завершении карьеры киберспортсмена, как вдруг ему позвонил товарищ по команде и сообщил радостное известие: организаторы финала предоставили команде Васи Wild Card (дополнительное место) для участия в турнире. Но возникла другая проблема — до начала финала осталось не так уж много времени, Васе срочно нужно попасть на вокзал и прибыть к месту проведения соревнования. Самый быстрый способ добраться до вокзала — воспользоваться местным метрополитеном. Вася забежал в метро и начал искать возможность приобрести билет для проезда. К счастью, на станциях метро недавно были установлены автоматы по продаже билетов. Автомат действует следующим образом. Сначала автомат принимает некоторое количество монет. Если этого количества недостаточно, то автомат ничего не делает (получить монеты назад невозможно). Если монет достаточно для покупки одного билета, то автомат рассчитывает размер сдачи. Если автомат может выдать требуемую сдачу (пользуясь уже находящимися в нем монетами), то он выдает билет и сдачу. В противном случае, он переходит в состояние ошибки и ничего не выдает, даже билет.

Однако вернемся к Васе. Вася собрался бросить несколько монет в автомат, но задумался — не случится ли так, что он останется без билета? Зная наборы монет, находящиеся в автомате и у Васи, определите, может ли Вася бросить в автомат часть своих монет (возможно — все монеты) так, что их будет достаточно для покупки билета, но автомат не сможет выдать сдачу.

При выдаче сдачи автомат также может использовать монеты, которые в него бросил Вася. Гарантируется, что у Васи хватает денег на покупку билета.

Формат входного файла

В первой строке даны три натуральных числа: N — количество монет у Васи, K — количество монет в автомате к моменту прихода Васи и M — стоимость билета метро ($1 \leq N, K, M \leq 5\,000$).

Во второй строке даны N натуральных чисел — номиналы монет Васи.

В третьей строке даны K натуральных чисел — номиналы монет, находящихся в автомате.

Монеты по номиналу не превосходят 10 000. Наборы монет заданы в неубывающем порядке по номиналу.

Формат выходного файла

Если Вася сможет получить билет, бросив в автомат любую сумму, достаточную для покупки жетона, то выведите слово **Go**. Если же Вася может перевести автомат в состояние ошибки, то выведите слово **Wait**.

Пример

stdin	stdout
3 2 4 1 2 3 1 2	Go
3 2 4 1 2 3 2 4	Wait

Разбор задачи D. Wild Card: Subway

Первое, что нужно заметить, Васе не имеет смысла бросать в автомат такое множество монет, что его подмножества достаточно для покупки билета, поскольку автомат сможет сразу сдать «лишние» монеты. Таким образом, Вася может использовать множества монет с суммой не более $M + A$, где A — номинал наибольшей монеты во множестве.

Найдем, какие суммы S в интервале от M до $M + A$ может получить Вася, используя свои монеты. Для этого воспользуемся динамическим программированием. Значение $d[i, s]$ будет равно 1, если Вася может получить сумму ровно s , используя подмножества первых i монет и 0 — в противном случае. Очевидно, что $d[0, 0] = 1$ и для любого $s > 0$ $d[0, s] = 0$. Далее для любого i от 0 до $N - 1$ выполним процедуру заполнения следующей строки матрицы d . Для всех чисел s от 0 до $M + A$ выполняется: если $d[i, s] = 1$, то $d[i + 1, s] = 1$ и $d[i + 1, s + A[i]] = 1$, где $A[i]$ — номинал монеты i . Вася может получить сумму s если $d[N, s] = 1$. Аналогично можно найти множество SA сумм, которые может сдать автомат, используя свои монеты. Соответственно, если для некоторого x выполняется условие: если $x + M$ принадлежит S , но не принадлежит SA , то нужно выводить «Wait». В противном случае нужно вывести «Go».

Задача E. Wild Card: Bus

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Не сумев приобрести билет в метро (см. задачу «Wild Card: Subway»), Вася решил воспользоваться автобусом для поездки на вокзал. Город, в котором живет Вася, представляет собой множество остановок и двусторонних дорог, связывающих остановки. Автобус может перемещаться по дорогам между остановками. Для каждой дороги известно, сколько времени требуется автобусу для преодоления ее. Кроме того, автобус может находиться на остановке произвольное неотрицательно количество времени.

Вася уже давно играет в Dota2, благодаря этому он стал известным человеком в своем городе. Один из фанатов Васи, Петя, утверждает, что сумел получить автограф Васи на одной из остановок города. Точное время получения автографа Петя не помнит, но утверждает, что это произошло в какой-то момент времени в интервале $[t_b, t_e]$. Не все жители города поверили словам Пети. Для того, чтобы опровергнуть слова Пети, некоторые даже стали изучать показания камер наблюдения, установленных на некоторых остановках. Зная времена, требуемые для проезда между остановками, и показания камеры наблюдения определите, мог ли Вася дать автограф Пете на указанной остановке в предполагаемый интервал времени.

Формат входного файла

В первой строке находятся два числа N, M — количество остановок в городе и количество дорог соответственно $\left(1 \leq N \leq 1\,000, 0 \leq M \leq \frac{N(N-1)}{2}\right)$.

В следующих M строках описывается карта города. В каждой строке даны числа a_i, b_i, l_i ($1 \leq a_i < b_i \leq N, 1 \leq l_i \leq 10\,000$) — номера остановок, соединенных дорогой, и время, нужное автобусу для перемещения по этой дороге. Каждая пара остановок соединена не более чем одной дорогой. По любой дороге можно двигаться в обе стороны.

В следующей строке записаны три целых числа C, t_b и t_e ($1 \leq C \leq N, 0 \leq t_b \leq t_e \leq 15\,000\,000$) — номер остановки, указанной Петей, самое раннее и самое позднее время (включительно), в которое Вася мог оставить автограф.

В следующей строке находится одно число K ($1 \leq K \leq 50\,000$) — количество показаний с камер наблюдения.

В следующих K строках записаны пары целых чисел p_j, t_j ($1 \leq p_j \leq N, 0 \leq t_j \leq 15\,000\,000$) — номер остановки и время, когда ка-

мера зафиксировала Васю, едущего в автобусе. Все временные отметки t_j различны. Водитель автобуса не так торопится на вокзал, как Вася, поэтому его путь не является оптимальным, более того, автобус может проезжать по одним и тем же дорогам несколько раз в процессе одного рейса.

Формат выходного файла

Если Вася мог дать автограф Пете в указанный интервал времени на указанной остановке, то выведите YES, в противном случае — выведите NO.

Пример

stdin	stdout
4 6 1 2 10 2 3 10 3 4 10 1 4 10 1 3 15 2 4 15 2 12 25 2 1 10 3 30	YES
4 6 1 2 10 2 3 10 3 4 10 1 4 10 1 3 15 2 4 15 2 12 15 2 1 10 3 25	NO

Разбор задачи E. Wild Card: Bus

Теоретически Вася мог побывать на остановке C в интервал времени между двумя показаниями камеры, либо после последнего показания, либо перед первым показанием. Пусть даны два соседних показания камеры. В первом говорится, что Вася был на остановке a_1 в момент времени t_1 , во вто-

ром — то что Вася был на остановке a_2 в момент времени t_2 . Для того, чтобы добраться до остановки C из a_1 и перейти в a_2 Васе понадобится время $t = d[a_1, c] + d[c, a_2]$, где $d[i, j]$ — длина кратчайшего пути между остановками i и j . Если $t \leq t_2 - t_1$, то Вася мог быть на остановке C в интервал времени $(t_1 + d[a_1, c], t_2 - d[c, a_2])$. Аналогичная ситуация будет с посещением города C перед первым показанием камеры и после последнего показания камеры, только в этих случаях одна из границ интервала будет бесконечной. Поскольку дороги двунаправленные, то $d[a_1, c] = d[c, a_1]$. Значит, для получения времени t достаточно знать расстояния от перекрестка C до всех остальных перекрестков. Это можно сделать за $O(N^2)$, запустив алгоритм дейкстры из перекрестка C .

Получив множество интервалов времени необходимо проверить, пересекается ли хотя бы один из них с исходным интервалом (t_b, t_e) и вывести «YES» в случае пересечения.

Задача F. Windrunner at Your Service

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася тренируется играть в Dota2. Сейчас он играет за героя Windrunner. Одна из способностей героя — Shackleshot — позволяет привязать противника к дереву. Для осуществления этого действия необходимо выполнение следующих условий:

- расстояние от героя до врага не должно превышать $D1$
- расстояние от врага до дерева не должно превышать $D2$
- угол между векторами «герой–враг» и «враг–дерево» не должен превышать A градусов по абсолютному значению.

Вася знает положение своего героя (X_w, Y_w) , положение врага (X_e, Y_e) , а так же положение всех N деревьев на карте (X_i, Y_i) . Считается, что враг неподвижен. Помогите Васе определить, какое наименьшее расстояние придется преодолеть его герою для того, чтобы привязать врага к одному из деревьев.

Формат входного файла

Первая строка входного файла содержит три целых числа — $D1, D2$ и A ($1 \leq D1, D2 \leq 1000, 1 \leq A \leq 90$). Во второй строке содержатся четыре целых числа — X_w, Y_w, X_e, Y_e . В третьей строке находится единственное натуральное число N ($1 \leq N \leq 1000$) — количество деревьев расположенных на

карте. В следующих N строках находятся координаты деревьев — X_i, Y_i . Все координаты (героя, врага и деревьев) являются целыми числами и не превосходят 1000 по абсолютному значению. Ни одно дерево не находится в одной точке с вражеским героем.

Формат выходного файла

Если Вася может привязать врага к какому-либо дереву, то необходимо вывести минимальный путь, который придется проделать его герою, с абсолютной или относительной погрешностью не более 10^{-6} . В противном случае необходимо вывести -1 .

Пример

stdin	stdout
1 2 30 0 0 1 1 1 2 2	0.4142135624
1 2 30 0 0 1 1 1 3 3	-1
2 2 30 0 0 1 1 1 2 2	0.0000000000

Разбор задачи F. Windrunner at Your Service

Каждое дерево, находящееся на расстоянии не более D_2 от врага, дает сектор круга, из которого можно выполнять привязку. Радиус сектора — D_1 , а центр находится в точке положения врага. Для решения задачи необходимо найти наименьшее расстояние от положения героя до одного из секторов.

Если герой уже находится в каком-то секторе, то ответ на задачу будет 0. Легко проверить, что если угол между направлением «герой-враг» и направлением «враг-дерево» удовлетворяет условию (не больше A), но герой находится на расстоянии $D > D_1$ от врага, то расстояние до сектора будет равно $D - D_1$.

Так же легко проверить, что если исходный угол больше A , то ближайшая к герою точка на секторе круга будет лежать на одном из двух граничных отрезков-радиусов сектора. Тогда расстояние до сектора будет равно наименьшему расстоянию до этих отрезков.

Задача G. What is the Answer?

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася хорошо научился играть в Dota2. Теперь Вася решил участвовать в Challenge 24 — популярном соревновании по программированию. На отборочном раунде ему встретилась задача, очки за которую начисляются в зависимости от того, насколько эффективно решение участника по отношению к эффективности лучшего отправленного кем-либо решения. Эффективность решения оценивается одним натуральным числом A — абсолютным показателем решения. Чем меньше это число, тем лучше решение. Баллы за задачу (относительный показатель решения) начисляются по следующей формуле:

$$R = [100 * (1 - \sqrt{(1 - BA/A)})]$$

где BA — абсолютный показатель лучшего отправленного решения, а A — абсолютный показатель решения участника, а $[X]$ — целая часть числа X . В частности, из формулы следует, что если участник отправил наилучшее решение, то он получит 100 баллов.

По правилам соревнования, абсолютный показатель наилучшего решения не сообщается до конца раунда. Однако, каждому участнику сообщается абсолютный и относительный показатель его решения. Вася отправил решение с абсолютным показателем A и получил R баллов (относительный показатель). Помогите Васе узнать наименьшее и наибольшее возможные значения абсолютного показателя наилучшего решения.

Формат входного файла

В единственной строке входного файла содержатся два натуральных числа — A и R , $1 \leq A \leq 10^9$, $1 \leq R \leq 100$.

Формат выходного файла

Если ни при каком значении абсолютного показателя наилучшего решения Вася не мог получить такие баллы, то выведите -1 . В противном случае необходимо вывести 2 числа — наименьший и наибольший возможные абсолютные показатели наилучшего решения.

Пример

stdin	stdout
100 99	-1
100 50	75 75
100 10	19 20

Разбор задачи G. What is the Answer?

Пусть R_1 — неокругленное число очков, которое получил участник, т.е. $R_1 = 100 * (1 - \sqrt{1 - BA/A})$ и известно, что $R \leq R_1 < R$.

Перепишем формулу из условия так, чтобы BA было выражено через R_1 и A . Получим:

$$BA = (10^4 - (10^2 - R_1)^2) * A / 10000 \quad (*)$$

Если мы подставим вместо R_1 число R и округлим вверх значение выражения (*), то мы получим наименьшее возможное значение BA . Аналогично, если мы подставим вместо R_1 значение $R + 1$, посчитаем значение выражения (*) и возьмем наибольшее целое число, меньшее этого значение, то мы получим наибольшее возможное значение BA . Соответственно, если наибольшее полученное значение меньше наименьшего, то ответ на задачу — “-1”.

Задача H. We Admit No Defeat

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

Вася тренируется играть в Dota2. Сейчас он играет за героя Mirana. Впрочем, в данной задаче это неважно — герои не будут пользоваться своими специальными способностями. Вася предстоит сразиться один на один с вражеским героем. Разумеется, Вася не приемлет поражение! Битва будет честной, поэтому герои будут просто наносить удары друг другу. Можно считать, что скорость атаки героев одинакова и они наносят удары одновременно. Герои будут наносить удары, пока один из них (или оба) не погибнет. Несмотря на огромное желание победить, герои подчиняются игровой механике. Для поединка важны следующие параметры героев:

- Количество очков здоровья
- Базовый урон

- Броня

При нанесении удара урон считается по следующей формуле $D = BD / (1 + A * 0.06)$, где D — урон, нанесенный герою, BD — базовый урон атакующего героя, A — броня героя, получившего удар. Итоговое значение урона не округляется и вычитается из количества очков здоровья героя, получившего удар. Если количество очков здоровья в результате удара станет неположительным, то считается, что герой убит. Поскольку два героя атакуют одновременно, то может возникнуть ситуация, при которой оба героя погибают. Базовый урон каждого героя строго положителен, поэтому битва будет конечной.

Кроме того, герои могут покупать предметы, которые увеличивают одну (или несколько) характеристики. Каждый герой может купить максимум 6 предметов, один и тот же предмет можно покупать несколько раз. После того, как предметы куплены, начинается битва. Помогите Васе определить, сможет ли он выжить в битве?

Формат входного файла

Первая строка входного файла содержит три числа — начальное количество очков здоровья, базовый урон и броня героя Васи. Во второй строке приводится описание вражеского героя в том же формате. В следующей строке находится единственное число N ($1 \leq N \leq 20$) — количество возможных предметов для покупки. Далее идет описание предметов. Первая строка описания содержит название предмета (непустая строка, состоящая не более, чем из 50 букв латинского алфавита) и число K ($1 \leq K \leq 3$) — количество бонусов от предмета. В следующих K строках находится описание бонусов. Каждое описание бонуса имеет вид:

`<BonusName> +X`

где `<BonusName>` — тип изменяемой характеристики:

- 'HP' — для количества очков здоровья
- 'Damage' — для базового урона
- 'Armor' — для брони

X — численное значение бонуса к характеристике. Все бонусы от одного предмета имеют разные значения `<BonusName>`. Все характеристики (как начальные характеристики героев, так и бонусы предметов) — целые неотрицательные числа, не превосходящие 1000. Бонусы от предметов не могут быть нулевыми. Начальное здоровье героя строго положительно. Все названия предметов различны. Смотрите пример для уточнения формата входных данных.

Формат выходного файла

Если для любого множества купленных вражеским героям предметов существует множество предметов, при покупке которых герой Васи останется жив, то нужно вывести 1. Если существует множество предметов, при покупке которых вражеским героям герой Васи умрет независимо от того, какие предметы купит ему Вася, то следует вывести -1 .

Пример

stdin	stdout
600 40 1 450 40 8 1 IronBranch 2 Damage +1 HP +19	-1
600 40 1 450 40 8 2 IronBranch 2 Damage +1 HP +19 Chainmail 1 Armor +5	1

Разбор задачи H. We Admit No Defeat

Для определения выживет ли герой в бою важно значение его характеристик (H — “HP”, D — “Damage”, A — “Armor”) и значение характеристик врага. Заменим значение величины здоровья H на выражение $AH = H * (1 + 0.06 \times A)$ (“полное здоровье”) это будет эквивалентно тому, что при уроне D значение AH будет уменьшаться ровно на D (без учета брони).

Всего для каждого героя возможных вариантов покупки 6-ти предметов — 6^N . Однако, заметим, что значение величины D при любом множестве купленных предметов не превосходит 7000 ($1000 + 6*1000$). Достаточно определить, какое наибольшее значение полного здоровья $MaxAH[d]$ может получить герой для каждого возможного значения урона d . Для этого переберем все 6^N вариантов покупки предметов, для каждого варианта посчитаем значения d и AH и обновим значение $MaxAH[d]$ если необходимо.

Таким образом, мы получим максимум 7000 вариантов оптимальной покупки предметов для каждого из героев. Перебрав все пары таких вариантов определим, верно ли, что Вася сможет выжить в битве при любом наборе предметов вражеского героя.

Задача I. Wex-Wex-Wex

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася играет в Dota 2. Одним из самых сложных и интересных героев в игре является Invoker. Как и любой другой настоящий герой, Invoker должен убивать! В ряд перед героем выстроились N крипов (неуправляемых юнитов), у каждого из которых ровно H единиц здоровья. Крипы расположены в точках с координатами от 1 до N , по одному крипу в каждой целочисленной точке. Invoker может убивать крипов при помощи своих заклинаний. На самом деле, он может использовать 10 различных заклинаний, но в данной задаче он будет использовать только 3 из них.

- Chaos Meteor — наносит одинаковый фиксированный урон всем крипам на некотором отрезке.
- Sun Strike — наносит фиксированный урон, который делится поровну на всех *живых* крипов на некотором отрезке. Если при делении урона получается нецелое число, то оно округляется вниз до целого.
- Tornado — поднимает крипов в воздух на некоторое время, делая их неуязвимыми к заклинаниям, после чего опускает на землю и наносит одинаковый фиксированный урон каждому из поднятых крипов. Как и предыдущие заклинания, это заклинание действует на некотором отрезке.

Крип считается мертвым, если количество единиц его здоровья после очередного заклинания стало не больше 0. Поднятые в воздух заклинанием Tornado крипы не учитываются при расчете урона от заклинания Sun Strike. Если заклинание Tornado заканчивает свое действие в момент времени T и в этот же момент времени используется другое заклинание, то сначала наносится урон от Tornado, а потом начинается действие другого заклинания.

По заданному набору использованных заклинаний определите, какое суммарное количество урона мог нанести Invoker по крипам.

Формат входного файла

В первой строке даны три натуральных числа N — количество крипов и H — начальное количество единиц здоровья каждого из крипов и M — количество примененных заклинаний. $1 \leq N, M \leq 100\,000$, $1 \leq H \leq 1\,000\,000\,000$.

В следующих M строках описаны примененные заклинания, по одному в строке в порядке их использования героем. Описание заклинания имеет вид:

$\langle \text{Type} \rangle \langle \text{Time} \rangle \langle L \rangle \langle R \rangle \langle D \rangle [\langle T \rangle]$, где

$\langle \text{Type} \rangle$ — тип примененного заклинания (1 — Chaos Meteor, 2 — Sun Strike, 3 — Tornado);

$\langle \text{Time} \rangle$ — время использования заклинания;

$\langle L \rangle$ — левая граница действия заклинания (включительно);

$\langle R \rangle$ — правая граница действия заклинания (включительно);

$\langle D \rangle$ — урон заклинания (при использовании Sun Strike делится между всеми живыми крипами);

$\langle T \rangle$ — время действия (только при заклинании Tornado);

$1 \leq \text{Time}, T \leq 1\,000\,000\,000$; $1 \leq L \leq R \leq N$; $1 \leq D \leq 1\,000\,000\,000$.

Заклинания затрагивают всех крипов на отрезке от $\langle L \rangle$ до $\langle R \rangle$ включительно. Все переменные времени $\langle \text{Time} \rangle$ различны. Гарантируется, что в любой момент времени «активно» не более одного Tornado.

Формат выходного файла

Вывести единственное число — суммарный урон, нанесенный героем.

Пример

stdin	stdout
5 10 7 1 1 2 5 4 1 2 2 2 8 2 3 1 3 9 3 4 1 4 2 2 2 5 3 5 4 2 6 3 5 4 1 7 5 5 6	44

Разбор задачи I. Wex-Wex-Wex

Эту задачу легко решить при помощи корневой декомпозиции. Разобьем множество из N крипов на \sqrt{N} отрезков по \sqrt{N} крипов в каждом. Для каждого отрезка будем хранить дополнительный урон, нанесенный одновременно

всем крипам на отрезке. Для каждого крипа будем хранить его текущее количество здоровья (без учета дополнительного урона). Кроме того, для каждого отрезка будем хранить список (массив) живых крипов этого отрезка, отсортированный по убыванию текущего количества единиц здоровья крипов и, собственно, само количество живых крипов.

Для решения задачи нам необходимо уметь быстро (за асимптотику $O(\sqrt{N})$) обрабатывать следующие запросы:

1. Нанесение фиксированного урона по всем крипам на произвольном интервале (L, R)
2. Нахождение количества живых крипов на произвольном интервале (L, R)

Пусть наносится урон D на интервале (L, R) . Переберем все наши отрезки. Если отрезок полностью попадает в интервал, то необходимо увеличить значение дополнительного уровня для этого отрезка. Кроме того, некоторое количество крипов этого отрезка может умереть в результате действия заклинания — значит, возможно придется сдвинуть указатель на последнего живого крипа (всего таких операций будет не более N для всех запросов, так как для каждого крипа указатель сдвигается максимум один раз). Если отрезок частично пересекается с интервалом (L, R) , то нанесем урон каждому крипу отдельно. При этом мог нарушиться порядок живых крипов в этом отрезке (так как только часть из них получила урон). Однако, поскольку относительный порядок крипов, получивших урон, и крипов, не получивших урона, не изменился, то восстановить общий порядок можно сделав сортировку слиянием для этих двух множеств. Итого, запрос будет обработан за $O(\sqrt{N})$.

Для подсчета количества живых крипов на интервале воспользуемся аналогичным подходом. Если отрезок полностью попадает в указанный интервал, то количество живых крипов в нем уже подсчитано. Если отрезок попадает в интервал частично, то найти количество живых крипов на пересечении можно просто проверив каждого крипа отдельно.

Теперь можно легко обработать исходные запросы в задаче. Заклинание “Chaos Meteor” наносит фиксированный урон, поэтому достаточно выполнить первую операцию. Для заклинания “Sun Strike” сначала необходимо найти количество живых крипов при помощи второй операции, а затем нанести фиксированный урон при помощи первой операции. Для заклинания “Tornado” запомним границы действия. Для каждого последующего заклинания, применяемого во время действия Tornado, исключим из интервала применения интервал действия Tornado. При этом может получиться 0, 1 или 2 новых отрезка. Урон на них наносится независимо, а количество живых крипов на сумме отрезков — сумма запросов к каждому из отрезков. После того, как Tornado

закончит действовать, необходимо нанести фиксированный урон при помощи первой операции.

Таким образом, каждый из запросов можно обработать за $O(\sqrt{N})$.

Задача J. Warlock-2 (Высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася тренируется играть в Dota2. Сейчас он играет за героя Warlock. Одной из способностей героя является Fatal Bounds — герой «связывает» несколько вражеских юнитов вместе, заставляя их получать дополнительный урон. Заклинание работает следующим образом: если один из связанных юнитов получает урон (основной урон), то все остальные юниты получают P процентов от полученного юнитом урона (дополнительный урон). Действие связи не распространяется на дополнительный урон, полученный в результате действия заклинания (дополнительный урон не вызывает получение дополнительного урона остальными юнитами). Юнит, получивший основной урон, не получает дополнительный урон. Если юнит погибает при получении основного урона (количество очков жизни становится неположительным), для дополнительного урона будет учитываться только количество единиц жизни юнита перед получением основного урона.

Вася хочет убить вражеского героя, поэтому связал его с N крипами при помощи Fatal Bounds. Далее будем считать, что вражеский герой не получает основной урон, в то время как все крипы будут воевать с Васей, пока не будут убиты. При этом часть полученного ими урона будет переходить Васе как дополнительный урон. Разумеется, сами крипы тоже будут страдать от получения дополнительного урона от Fatal Bounds. Вася может атаковать крипов произвольное число раз, при этом каждый раз он выбирает одного живого крипа, которому будет наноситься урон, и величину урона (положительное число). Вася атакует крипов, пока они не будут убиты. Зная изначальное количество очков здоровья у крипов, определите, какой наименьший и какой наибольший урон может при этом получить вражеский герой.

Формат входного файла

В первой строке входного файла содержатся два целых числа: N — количество крипов, связанных с вражеским героем, $1 \leq N \leq 100000$ и P — процент дополнительного урона относительного основного, $0 \leq P \leq 100$. Во

второй строке содержатся N натуральных чисел, не превосходящих 1000 — количество единиц здоровья у крипов, связанных Fatal Bounds.

Формат выходного файла

Необходимо вывести два вещественных числа с абсолютной или относительной погрешностью не более 10^{-6} — наименьший и наибольший урон, который может получить вражеский герой.

Пример

stdin	stdout
2 20	20.0000000000
100 10	21.6000000000

Разбор задачи J. Warlock-2 (Высшая лига)

Нахождение наибольшего возможного урона описано в разборе задачи «C. Warlock». Из аналогичных утверждений следует, что для получения наименьшего урона необходимо, чтобы в каждый момент нанесения урона было как можно больше живых крипов. Поэтому для оптимальной стратегии достаточно наносить урон крипу с текущим наибольшим количеством здоровья. При этом, количество его здоровья уменьшается быстрее, чем у остальных крипов (при $P < 100$), таким образом, он может сравняться по здоровью с другим крипом. В общем случае в каждый момент времени мы будем рассматривать множество крипов, обладающих наибольшим здоровьем, крипа, обладающего следующим по величине здоровьем, и крипа, обладающего наименьшим количеством здоровья.

Если у нас имеется группа из K крипов, обладающих одинаковым здоровьем, и мы хотим нанести единицу урона, то оптимально наносить урон по-ровну между крипами этой группы. При этом каждый крип в группе получит $X = 1/K + (K - 1)/K \times P$ единиц урона. Все остальные крипы получат P единиц урона.

В процессе нанесения урона у нас будут происходить два вида событий.

1. Группа сравняется по количеству здоровья со следующим крипом. В этом случае его надо добавить в группу увеличить размер группы на 1) и перейти к следующему по здоровью крипу.
2. Крип с наименьшим здоровьем умрет. В этом случае необходимо перейти к следующему крипу с наименьшим здоровьем среди живых оставшихся.

Для того, чтобы произошло первое событие, необходимо нанести $D_1 = DH/(X - P)$ единиц урона, где DH — разница в количестве здоровья у

крипов группы и следующего крипа. Если $X \leq P$ (крип умирает быстрее, чем группа), то такое событие не произойдет. Для второго события необходимо нанести $D_2 = H_{min}/P$ единиц урона, где H_{min} — здоровье “наименьшего” крипа. Далее нужно нанести $\min(D_1, D_2)$ урона и обработать соответствующее событие.

Эти события надо обрабатывать до тех пор, пока не останется одна группа из K крипов со здоровьем H у каждого. Чтобы “оптимально” убить эту группу, нужно нанести еще H/X единиц урона. Соответственно, герой получит P процентов урона от нанесенного в процессе убийства всех крипов.

Задача K. World of Dota: Cross 2 (Высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася играет в Dota 2, но ему надоело воевать каждый раз на одной и той же карте. Поэтому Вася решил сделать новую карту для этой игры. Для простоты будем полагать, что карта представляет собой прямоугольник $N \times N$, каждая клетка которого может быть либо свободна, либо занята (деревом или водой). Вася хочет, чтобы вражеская команда (состоящая из пяти героев) не могла группироваться близко к одной точке. Поэтому на карте не должно быть пяти свободных клеток, образующих крест с шириной и высотой 3:

```
. * .  
* * *  
. * .
```

Для того, чтобы предотвратить появление таких крестов, Вася может сажать деревья в свободные клетки. Кроме того, вдоль диагонали карты течет вода. Это значит, что каждая клетка главной диагонали исходного прямоугольника $N \times N$ занята водой. Сажать деревья на воду нельзя, но и вражеские герои на ней не располагаются.

Вася хочет оставить как можно больше пространства для маневров, поэтому деревьев на карте должно быть как можно меньше. Вася уже расставил несколько деревьев на карте. Помогите Васе расставить еще как можно меньше деревьев так, чтобы пять вражеских героев не могли образовать крест.

Формат входного файла

В первой строке находится единственное натуральное число — N — размер карты ($1 \leq N \leq 17$). Далее следуют N строк по N символов в каждой —

описание карты. Символ '.' соответствует свободной клетке, символ '#' — занятой (деревом или водой). Гарантируется, что все клетки главной диагонали заняты.

Формат выходного файла

Необходимо вывести описание новой карты — N строк по N символов в каждой. Если клетку следует оставить свободной, то надо выводить символ '.', если клетка изначально была занята — символ '#', если в изначально свободную клетку нужно добавить дерево — символ '+'. Если оптимальных решений несколько, можно вывести любое из них.

Пример

stdin	stdout
6	#.....
#.....	.#..+.
.#....	..#...
..#...	...#..
...#..	.+..#.
....#.#
.....#	
6	#.....
#.....	.#..+.
.#....	..#...
..#...	...#..
...#..	.#..#.
.#..#.#
.....#	

Разбор задачи K. World of Dota: Cross 2 (Высшая лига)

Наиболее просто решить данную задачу, используя перебор с отсечениями.

Клетку доски будем называть “активной”, если можно поставить крест с центром в этой клетке на доску. Определим множество изначально активных клеток и будем его поддерживать в процессе перебора. Очевидно, что пока остается хотя бы одна активная клетка, то нам нужно посадить еще хотя бы одно дерево. Рассмотрим первую активную клетку. Для того, чтобы запретить постановку креста в нее (сделать эту клетку “неактивной”) есть 5 вариантов постановки дерева (в эту клетку или в одну из соседних с ней по стороне). Переберем, в какую из 5 клеток поставить дерево, обновим множество активных

клеток и рекурсивно вызовем процедуру расстановки оставшихся деревьев. Итого, сложность алгоритма будет порядка 5^K , где K — ответ на задачу (минимальное число деревьев).

Этот алгоритм можно улучшить. Отсортируем все активные клетки (в порядке возрастания номера строки, при равенстве номеров строки — по номеру столбца). Тогда при обработке очередной активной клетки не имеет смысла ставить дерево в клетку над ней или в клетку левее нее. Таким образом, на каждом шаге достаточно перебирать всего 3 варианта постановки нового дерева. Кроме того, каждое дерево делает максимум 5 клеток неактивными, поэтому можно добавить следующее отсечение: если $K + (M + 4)/5 \geq A$, то нужно выйти из этой ветки перебора, где K — текущее количество уже поставленных деревьев, M — количество активных клеток, A — количество деревьев в текущем лучшем найденном решении. Такое отсечение позволяет значительно ускорить перебор, и решение будет укладываться в ограничение по времени.

Задача L. Wild Card: Plane (Высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася успешно добрался до вокзала, сел на аэроэкспресс и прибыл в аэропорт. Теперь Вася готов лететь на финал Starladder 2014. Однако, еще одно неприятное событие может помешать Васе прибыть на финал вовремя. Давным давно потухший вулкан вдруг проснулся и начал выбрасывать в воздух потоки золы и дыма. Несколько самолетов уже упали в непосредственной близости от вулкана, потеряв управление из-за вулканических пород, оказавшихся в воздухе. Для безопасного полета, самолет должен всегда находиться на расстоянии не менее r (вдоль поверхности Земли) от вулкана. Таким образом, часть воздушного пространства в виде круга с центром в местоположении вулкана закрыта для перелетов. Считая, что Земля имеет форму шара радиуса R , а так же зная координаты вулкана, начальной и конечной точек полета, определите, какое наименьшее расстояние придется преодолеть самолету, чтобы безопасно осуществить свой рейс. Высотой полета самолета относительно Земли нужно пренебречь — считается, что самолет летит на малой высоте вдоль поверхности.

Формат входного файла

Первая строка содержит два целых числа — R и r — радиус Земли и

расстояние от вулкана до границы закрытой зоны. $1 \leq r, R \leq 10^9$, гарантируется, что площадь поверхности закрытой зоны не превышает половины площади поверхности Земли. Во второй строке даны координаты вулкана — Lat_v и $Long_v$ — широта и долгота в градусах ($-90 \leq Lat_v \leq 90$, $-180 \leq Long_v \leq 180$). В третьей и четвертой строках даны координаты начальной и конечной точек полета в том же формате. Координаты всех точек целые. Гарантируется, что точки взлета и посадки различны и не лежат внутри или на границе закрытой зоны.

Формат выходного файла

Необходимо вывести единственное число — наименьшую длину пути безопасного полета вдоль поверхности Земли.

Пример

stdin	stdout
10 1 0 0 0 90 90 0	15.7079632679
10 1 0 0 0 45 0 -45	15.8083481818

Разбор задачи L. Wild Card: Plane (Высшая лига)

Легко понять, что кратчайшая траектория — это либо дуга большой окружности, соединяющая две точки, либо две дуги больших окружностей, касательных к границе запрещенной зоны и часть границы запрещенной зоны. Поэтому данная задача сводится к поиску касательных к окружности на сфере.

Для поиска касательных воспользуемся стереографической проекцией. Данное отображение обладает следующими свойствами:

1. Окружность на сфере, не проходящая через “северный полюс”, переходит в окружность на плоскости.
2. Окружность, проходящая через “северный полюс”, переходит в прямую.
3. Окружность, проходящая через оба “полюса”, переходит в прямую, проходящую через начало координат.

Это отображение удобно тем, что оно позволяет свести задачу к стандартной задаче на плоскости. Чтобы найти касательную из точки на сфере к окружности, повернем всю систему так, чтобы точка оказалась на “южном полюсе”, тогда ее образ попадет в начало координат. Геодезические, проходящие через “южный полюс”, превращаются в прямые, проходящие через начало координат. Если при запретная зона попадает на “северный полюс”, то образом круга будет внешность окружности на плоскости. Очевидно, что в этом случае всегда можно двигаться по геодезической на сфере. В противном случае, запретная зона перейдет во внутренность круга. Таким образом, мы свели задачу к стандартной задаче поиска касательной к окружности на плоскости. При этом так же нужно найти образ второй точки, чтобы проверить, пересекает ли геодезическая на сфере запрещенную зону.

День четвёртый (18.02.2014 г.) Конкурс Ярослава Твердохлеба и Романа Едемского

Об авторах...

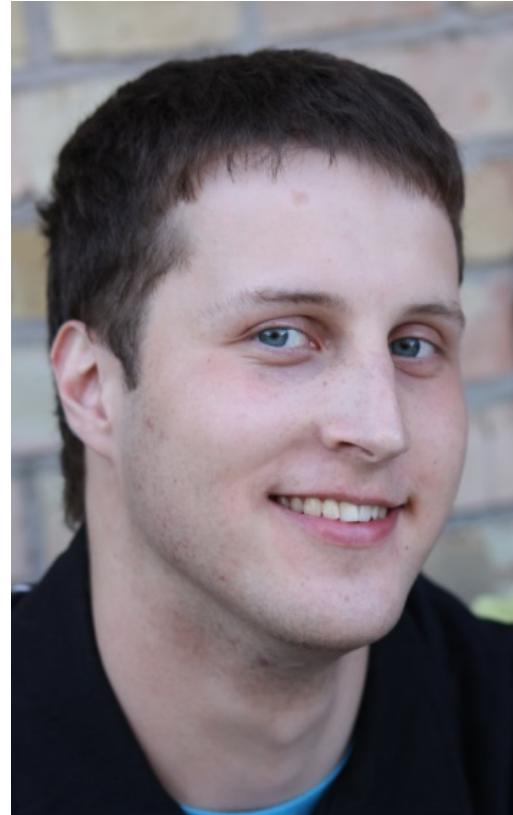
Ярослав Твердохлеб, родился 25 сентября 1992 года в городе Киеве. Учился в школе № 302 г. Киева с 1999–2005 г., в лицее № 171 “Лидер” с 2005–2009 г. В 2013 году получил диплом бакалавра по специальности “Прикладная математика” на факультете кибернетики Киевского национального университета имени Тараса Шевченко. С 2014 года учится в магистратуре факультета кибернетики.

Летом 2011 года стажировался в Google (Zurich), а летом 2012 года — в Facebook (Menlo Park, California).

Работает в киевском офисе Yandex.

Основные достижения:

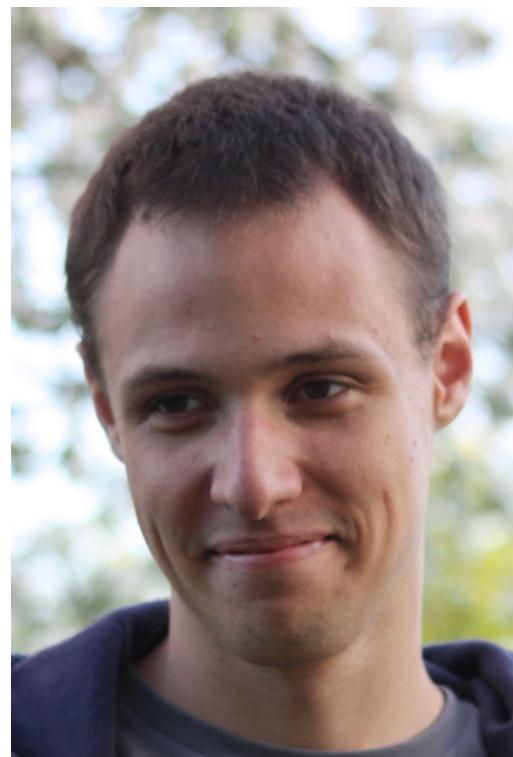
- Золотая медаль на IOI 2009 в (г. Пловдив, Болгария).
- 1 место на SEERC 2012 и 2 место на SEERC 2013.
- Серебряная медаль на ACM ICPC World Finals 2013 в составе команды Киевского национального университета (г. Санкт-Петербург, Россия).
- Финалист Yandex.Algorithm 2013 (5 место).
- 2 место на ABBYY Cup 3.0.
- Финалист Russian Code Cup 2013 (11 место).
- Финалист Facebook Hacker Cup 2013.
- Диплом первой степени (4 место) Snarknews Summer Series 2013.
- 3 место на Onsite раунде открытого кубка им. Е.В. Панкратьева в составе команды BZFlags.



Роман Едемский, родился 19 июля 1992 года в городе Киеве. Учился в “Британской Международной Школе” г. Киева с 1999–2003 г., в Техническом Лицее при НТУУ КПИ 2003–2005 г., в лицее № 171 “Лидер” с 2005–2009 г. В 2013 году получил диплом бакалавра по специальности “Прикладная математика” на факультете кибернетики Киевского национального университета имени Тараса Шевченка. С 2014 года учится в магистратуре факультета кибернетики.

Летом 2011 года стажировался в Google (Mountain View, California), а летом 2012 года — в Facebook (Menlo Park, California).

Работает в киевском офисе Yandex.



Основные достижения:

- Участник отборов на IOI 2009.
- 1 место на SEERC 2012 и 2 место на SEERC 2013.
- Серебряная медаль на ACM ICPC World Finals 2013 в составе команды Киевского национального университета (г. Санкт-Петербург, Россия).
- 3 место на Onsite раунде открытого кубка им. Е.В. Панкратьева в составе команды BZFlags.
- Автор задач для Всеукраинской олимпиады школьников по информатике.

Теоретический материал. Оптимизация динамического программирования за счет использования свойств линейных функций и алгоритма Грэхема

Постановка задачи 1

Рассмотрим следующую задачу. Есть N городов, расположенных на одной прямой. Город с номером i имеет координату x_i (координаты считаются начиная от какой-то точки на прямой и измеряются в километрах), причем координаты всех городов различны. Все города пронумерованы в порядке увеличения

их координат. Столица имеет номер 1. В каждом городе находится посланник, который в случае необходимости может доставить письмо в столицу. Каждый посланник характеризуется двумя числами: S_i и V_i . Здесь S_i — это время, необходимое посланнику для сборов в дорогу, а V_i — это время в минутах, за которое посланник i проходит один километр пути. Процесс доставки письма из города i ($i > 0$) в столицу выглядит следующим образом: посланник из города i собирается в течении S_i минут, после чего он двигается в город с номером $i - 1$. Затем, он либо продолжает движение без остановок, либо передает письмо посланнику, находящемуся в городе $i - 1$, после чего тот доставляет письмо по аналогичной схеме, тратя время на сборы и т.д.. В общем случае, в процессе доставки письма в столицу может принимать участие сколько угодно посланников. Требуется для каждого города вычислить наименьшее время, за которое письмо из него может быть доставлено в столицу.

Решение задачи 1

Будем решать задачу при помощи динамического программирования. Обозначим через $f(i)$ наименьшее время, необходимое для доставки письма из города i в столицу. Очевидно, $f(1) = 0$. Запишем формулу для вычисления $f(i)$ через уже вычисленные $f(j)$ для всех $j < i$.

$$f(i) = \min_{j < i} \{(x_i - x_j) \cdot V_i + S_i + f(j)\}$$

Приведенная рекуррентность уже позволяет решить задачу за $O(N^2)$. Для дальнейшей оптимизации раскроем скобки и вынесем из под минимума переменные, не зависящие от j :

$$f(i) = \min_{j < i} \{f(j) - x_j \cdot V_i\} + x_i \cdot V_i + S_i$$

Основная сложность заключается в том, чтобы вычислить минимум по j быстрее, чем за $O(N)$. Для этого дадим некую геометрическую интерпретацию этой формуле. Вспомним, что любую не вертикальную прямую можно задать уравнением вида $y = k \cdot x + l$, где k и l — это произвольные параметры, x — независимая переменная, а y — зависимая переменная. Число k еще называют угловым коэффициентом прямой. Итак, поставим в соответствие каждому городу j ($j < i$) прямую с $k = -x_j$ и $l = f(j)$. Мы можем такое сделать, поскольку для всех таких городов мы уже знаем значение $f(j)$. Тогда задача поиска минимума по j значения функции $f(j) - x_j \cdot V_i$ сводится к такой: есть набор прямых, заданных коэффициентами k и l , и требуется найти наименьшую y -координату пересечения прямых из этого набора с вертикальной прямой $x = V_i$. Для каждого x' выберем самую нижнюю точку пересечения вертикальной прямой $x = x'$ и прямых из нашего набора. Назовем получившееся множество точек нижним огибающим множеством для заданного набора прямых.

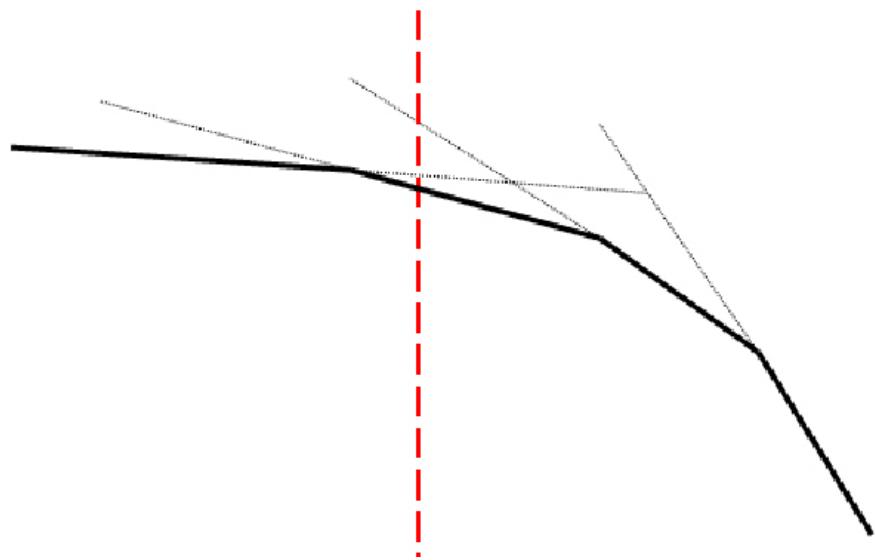


Рисунок 1.1 – Нижнее огибающее множество выделено черным

Что нам это дает? Как минимум то, что теперь нам уже не нужно пересекать вертикальную прямую со всеми прямыми из набора и выбирать самую нижнюю точку, а достаточно лишь пересечь вертикальную прямую $x = V_i$ с нижним огибающим множеством, после чего сказать, что ордината точки пересечения и есть искомым значением минимума по j .

Итак, если мы научимся поддерживать нижнее огибающее множество для набора прямых, а также быстро пересекать его с вертикальными прямыми, то мы сможем быстро вычислять $f(i)$ через $f(j)$ для $j < i$ и, как следствие, решать задачу быстрее, чем за $O(N^2)$. Для начала разберемся с тем, как такое множество поддерживать, а именно, как нужно перестроить нижнее огибающее множество после добавления в наш набор новой прямой (после того, как мы уже вычислили $f(i)$). Во-первых, заметим, что в силу специфики задачи, угловые коэффициенты всех прямых строго убывают при увеличении номера, поскольку координаты городов строго возрастают, а угловой коэффициент $k_i = -x_i$. Следовательно, при добавлении в набор новой прямой, ее часть точно войдет в нижнее огибающее множество, поскольку начиная с некоторого x именно эта новая прямая будет иметь наименьший y среди всего набора. В то же время, некоторые прямые уже не будут ничего вкладывать в нижнее огибающее множество и про них в дальнейшем можно забыть.

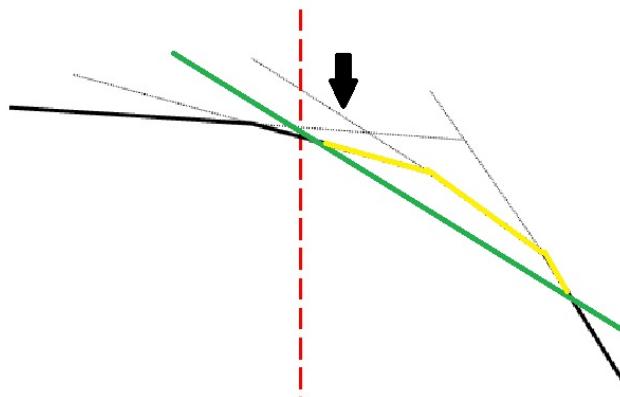


Рисунок 1.2 – Зеленым обозначена добавляемая прямая, желтым — та часть старого нижнего огибающего множества, которая больше ему не принадлежит, а стрелкой — прямая, которая теперь вообще не имеет пересечения с нижним огибающим множеством

Заметим, что нижнее огибающее множество состоит из отрезков и лучей, являющихся частями прямых из набора, причем угловые коэффициенты этих прямых убывают слева направо, а также каждая прямая вносит не более одного отрезка (или луча). Следовательно, для того, чтобы полностью задать нижнее огибающее множество, достаточно хранить номера прямых, части которых в него входят. Новая прямая всегда будет добавляться в конец этого массива, поскольку ее угловой коэффициент строго меньше, чем угловые коэффициенты прямых из нижнего огибающего множества. Тогда для того, чтобы добавить новую прямую в нижнее огибающее множество, выполним следующий алгоритм.

```
AddLine(L, S){  
    /* L --- добавляемая прямая, S --- массив  
    прямых, входящих в нижнее огибающее множество */  
    Пока в массиве больше одной прямой :  
        Взять последнюю прямую ( $L_1$ ) из S  
        Взять предпоследнюю прямую ( $L_2$ ) из S  
        Найти  $X$  точки пересечения  $x_1$  прямых  $L_1$  и  $L_2$   
        Найти  $X$  точки пересечения  $x_2$  прямых  $L_1$  и  $L$   
        Если  $x_1 < x_2$  :  
            | Остановить выполнение цикла  
        иначе  
            | Удалить прямую  $L_1$  из конца массива  
  
    }  
    Добавить прямую  $L$  в конец списка
```

Таким образом, мы последовательно будем удалять из конца массива те прямые, части которых больше не будут входить в нижнее огибающее множество, после чего добавим новую прямую в конец массива. Ясно, что каждая прямая будет удаляться из списка не более одного раза, поэтому в сумме мы потратим $O(N)$ операций на поддержание нижнего огибающего множества.

Итак, поддерживать нижнее огибающее множество мы научились, теперь осталось быстро находить его пересечение с вертикальной прямой. Это можно сделать при помощи бинарного поиска. Заметим, что точки пересечения соседних прямых в порядке их хранения в нашем списке отсортированы по возрастанию X координаты, поэтому мы можем делать бинарный поиск по ним для того, чтобы найти две точки соседние пересечения прямых из нижнего огибающего множества, которые лежат, соответственно, слева и справа от нашей вертикальной прямой. Теперь дело осталось за малым — подставить X координату вертикальной прямой в ту прямую, которая проходит через найденные 2 точки пересечения и найти Y координату пересечения вертикальной прямой и нижнего огибающего множества. Сложность нахождения этого пересечения — $O(\log N)$ на запрос. Следовательно, всю задачу мы теперь умеем решать за $O(N \log N)$.

Постановка задачи 2

Дан массив, состоящий из N целых положительных чисел. Также задана квадратичная функция $g(X) = a \cdot x^2 + b \cdot x + c$ своими коэффициентами a , b и c , причем $a < 0$. Требуется разбить массив на последовательные отрезки так, чтобы сумма значений g по всем отрезкам была максимальной, где под значением функции от отрезка подразумевается значение функции от суммы

чисел этого отрезка.

Решение задачи 2

Будем решать задачу при помощи динамического программирования. Обозначим через f_i наибольшую сумму значений функции g , если мы будем рассматривать (и разбивать на отрезки) только первые i элементов нашего массива. Будем считать, что $f(0) = 0$. Тогда можем записать формулу:

$$f(i) = \max_{j \leq i} \{a \cdot \text{sum}(j, i)^2 + b \cdot \text{sum}(j, i) + c + f(j - 1)\}$$

Под $\text{sum}(j, i)$ подразумевается сумма всех чисел на отрезке от j до i . Понятно, что подсчет такой динамики требует $O(N^2)$ операций. Для того, чтобы его ускорить, перепишем немного формулу, стоящую под минимумом, обозначив $\text{sum}(1, j)$ через $\sigma(j)$ ($\sigma(0) = 0$):

$$\begin{aligned} & f(j - 1) + a \cdot (\sigma(i) - \sigma(j - 1))^2 + b \cdot (\sigma(i) - \sigma(j)) + c = \\ &= f(j - 1) + a \cdot (\sigma(i)^2 - 2\sigma(i)\sigma(j - 1) + \sigma(j - 1)^2) + b \cdot (\sigma(i) - \sigma(j - 1)) + c = \\ &= (a\sigma(i)^2 + b\sigma(i) + c) + f(j - 1) - 2a\sigma(i)\sigma(j - 1) + a\sigma(j - 1)^2 - b\sigma(j - 1) \end{aligned}$$

Теперь запишем, чему равно $f(i)$:

$$f(i) = \max_{j \leq i} \{(a\sigma(i)^2 + b\sigma(i) + c) + f(j - 1) - 2a\sigma(i)\sigma(j - 1) + a\sigma(j - 1)^2 - b\sigma(j - 1)\}$$

Введем следующие обозначения: $t = (a\sigma(i)^2 + b\sigma(i) + c)$, $k = -2a\sigma(j - 1)$, $x = \sigma(i)$, $l = f(j - 1) + a\sigma(j - 1)^2 - b\sigma(j - 1)$. Тогда $f(i)$ выражается следующим образом:

$$f(i) = \max_{j \leq i} \{kx + l\} + t$$

Теперь у нас задача свелась к такой, которую мы уже умеем решать (см. предыдущую задачу из лекции). Единственное различие в том, что здесь нам нужно хранить верхнее огибающее множество, а не нижнее. То есть, мы можем решить эту задачу за $O(N)$.

Можно еще заметить, что поскольку $\sigma(i)$ монотонно возрастает при увеличении i , то абсцисса точки пересечения вертикальной прямой $x = \sigma(i)$ и верхнего огибающего множества будет двигаться только вправо. Следовательно, мы можем хранить указатель на текущий отрезок из верхнего огибающего множества, который пересекается с $x = \sigma(i)$ и двигать его при пересчете f для больших i . Таким образом, мы можем избавиться от бинарного поиска и добиться итоговой асимптотики решения $O(N)$.

Двойственная задача

Осталось сказать, что задача построения нижнего (верхнего) огибающего множества прямых $y = k_i \cdot x + l_i$ эквивалентна задаче построения нижней(верхней) цепи выпуклой оболочки точек (k_i, l_i) . Следовательно, вместо того чтобы пересекать прямые в процедуре перестройки огибающего множества, можно использовать векторные произведения как в алгоритме Грэхема для построения выпуклой оболочки. Отсюда и название лекции.

Задачи и разборы

Задача А. Счастливые пары (Юниорская лига)

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

Ксюша — счастливая обладательница массива a_i длины N , в котором ровно по одному разу встречаются числа $1, 2, \dots, N$. Ксюша считает пару индексов (i, j) счастливой в этом массиве, если одновременно выполняется $a_i < a_j$ и $i > j$. Посчитайте количество счастливых пар индексов в заданном массиве.

Формат входного файла

В первой строке задано одно число — N ($1 \leq N \leq 1000$). Во второй строке через пробел перечислены элементы массива a_i .

Формат выходного файла

Выведите количество счастливых пар индексов в заданном массиве.

Примеры

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

Разбор задачи А. Счастливые пары (Юниорская лига)

Необходимо было реализовать подсчет таких пар индексов i и j , что $i > j$ и $a_i < a_j$.

Асимптотика решения: $O(N^2)$.

Задача В. Лазер (Юниорская лига)

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

Сегодня в НИИ Ксюше, как самой ответственной сотруднице, поручили провести очень важный эксперимент. Суть опыта заключается в том, чтобы расплавить плитку шоколада с помощью новой лазерной установки. Установка состоит из прямоугольного ячеистого оперативного поля размером $n \times m$ сантиметров и руки-робота. Каждая ячейка поля представляет собой квадрат размера 1×1 . На руке-роботе расположены два лазера, направленные на поле перпендикулярно его плоскости. В любой момент времени лазеры попадают в центры некоторых двух клеток поля. Так как лазеры расположены на руке, то их движение происходит синхронно: при перемещении одного из лазеров на некоторый вектор, второй перемещается на тот же вектор.

Про эксперимент известно следующее:

- изначально оперативное поле полностью покрывает шоколадка размером $n \times m$, а также оба лазера находятся над полем и включены;
- шоколад, под воздействием лазера плавится только в пределах одной ячейки оперативного поля (над которой находится лазер), лазер должен попадать в центр ячейки поля;
- руку-робота можно передвигать только параллельно краям оперативного поля, после каждого перемещения лазеры должны указывать в центры ячеек;
- очень важно, чтобы в любой момент оба лазера находились над полем – Ксюша не хочет случайно сжечь лабораторию.

Даны n и m , а также ячейки (x_1, y_1) и (x_2, y_2) , над которыми изначально висят лазеры (x_i — номер столбца, y_i — номер строки). Будем считать, что строки нумеруются сверху вниз числами от 1 до m , а столбцы — числами от 1 до n слева направо. Необходимо найти количество ячеек оперативного поля, шоколад на которых не может быть расплавлен при соблюдении указанных правил.

Формат входного файла

В первой строке задано число t ($1 \leq t \leq 10000$) — количество наборов входных данных в тесте. В каждой из следующих

t строк перечислены через пробел целые числа n, m, x_1, y_1, x_2, y_2 ($2 \leq n, m \leq 10^9, 1 \leq x_1, x_2 \leq n, 1 \leq y_1, y_2 \leq m$). Ячейки (x_1, y_1) и (x_2, y_2) различны.

Формат выходного файла

В каждой из t строк выходного файла должен содержаться ответ на соответствующий набор входных данных.

Примеры

stdin	stdout
2	8
4 4 1 1 3 3	2
4 3 1 1 2 2	

Разбор задачи В. Лазер (Юниорская лига)

Пускай:

$$d_x = |x_1 - x_2| + 1, d_y = |y_1 - y_2| + 1,$$
$$I = \mu([1, n - d_x + 1] \cap [d_x, n]) \cdot \mu([1, m - d_y + 1] \cap [d_y, m]),$$

где $\mu([a, b]) = b - a + 1$.

Тогда ответ: $n \cdot m - 2 \times (n - d_x + 1) \times (m - d_y + 1) + I$.

Асимптотика решения: $O(1)$.

Задача С. Новогодние подарки (Юниорская лига)

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

Девочка Ксюша выбирает родителям новогодние подарки. В этом году она хочет подарить им набор новогодних шаров. Ксюша не хочет дарить набор шаров, составленный за нее профессиональными дизайнерами, поскольку лучший подарок — это тот, который человек делает (или собирает) собственно ручно. Поэтому, Ксюша решила собрать набор новогодних шаров в магазине “Гирлянды и туризм”.

В магазине “Гирлянды и туризм”, продаются **пронумерованные** шары раскрашенные в один из N цветов. Известно, что у всех новогодних шаров номера **различные**, а новогодних шаров каждого цвета одинаковое количество — K штук. Ксюша хочет купить набор из ровно N шаров, так чтобы, у нее было ровно по одному шару каждого цвета.

Сколько существует различных наборов из N шаров, которые Ксюша может подарить родителям? Ответ следует предоставить по модулю M .

Формат входного файла

Первая строка содержит число T ($1 \leq T \leq 1000$) — количество тестовых примеров. В каждой строке дано три числа: N, K и M ($1 \leq N, K, M \leq 10^{18}$).

Формат выходного файла

Для каждого тестового примера, в единственной строке необходимо вывести остаток по модулю M количества различных вариантов выбрать N шаров из заданных $N \cdot K$ таким образом, чтобы они были различных цветов. Учтите, что все шары уникальны ввиду поставленного на них номера.

Примеры

stdin	stdout
2	32
5 2 1000	25
2 5 1000	

Разбор задачи С. Новогодние подарки (Юниорская лига)

Нужно посчитать $K^N \bmod M$ при $1 \leq K, N, M \leq 10^{18}$. Для решения можно было:

1. Реализовать длинную арифметику (либо воспользоваться языком в котором она уже реализована) и написать алгоритм бинарного возведения числа в степень. Асимптотика решения: $O(\log N)$.
2. В python/python3 вызвать функцию $\text{pow}(K, N, M)$. Асимптотика решения: $O(\log N)$.
3. Реализовать алгоритм перемножения по модулю порядка 10^{18} по аналогии с алгоритмом бинарного возведения числа в степень. Асимптотика решения: $O(\log^2 N)$.

Задача D. Депутаты на дереве (Юниорская лига)

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

В следствии чрезмерной децентрализации, в стране X исчезла столица. Теперь в стране X сложная политическая ситуация, урегулировать которую можно лишь собрав новый совет депутатов.

В каждом из N городов страны X живут депутаты. В городе i ($1 \leq i \leq N$) живет ровно d_i депутатов. Между городами страны X проложено $N - 1$ двухсторонних дорог, по которым можно ровно одним путем добраться из каждого города в каждый другой. Для каждой дороги i ($1 \leq i \leq N - 1$) известны города, которые она соединяет u_i и v_i , а также стоимость проезда одного человека w_i по этой дороге.

Так как с финансами у страны X существуют серьезные проблемы, для спасения страны необходимо собрать совет депутатов в каком-то городе таким образом, чтобы суммарные затраты всех депутатов были наименьшими. Для упрощения задачи, вам как и.о. премьер-министра страны X разрешается объявить бесплатным проезд на $0 \leq K \leq N - 1$ дорогах.

Формат входного файла

В первой строке указаны два числа N и K ($1 \leq N \leq 1000$, $0 \leq K \leq N - 1$). Во второй строке расположено N целых чисел от 1 до 100000 — количество депутатов, проживающих в i -ом городе. В оставшихся $N - 1$ строке описаны дороги в стране X. Каждая дорога задана в отдельной строке тремя числами: u_i , v_i и w_i ($1 \leq u_i, v_i \leq N$, $1 \leq w_i \leq 100000$). Гарантируется, что из каждого города по заданным дорогам можно добраться в любой другой.

Формат выходного файла

Выведите единственное число — минимальное количество денег, которое придется потратить депутатам, чтобы собраться на совет в один город.

Примеры

stdin	stdout
<pre>5 3 8 5 1 8 7 1 3 8 2 3 9 3 5 7 3 4 8</pre>	45

Разбор задачи D. Депутаты на дереве (Юниорская лига)

В этой задаче нужно было выбрать вершину в дереве, в которой дешевле всего собраться депутатам, с учетом того, что на K ребрах можно отменить плату за проезд. Переберем вершину, где будут собираться депутаты. Тогда, для каждого депутата ясно, по каким ребрам ему предстоит идти в вершину сбора. Подсчитаем для каждого ребра суммарное количество денег, которые заплатят депутаты проходя по нему. Зная персональный вклад в суммарный ответ для каждого ребра, можно определить K самых весомых ребер и удалить их. Таким образом, для каждой выбранной вершины сбора за время $O(N \log N)$ можно посчитать минимальную стоимость, которую необходимо уплатить депутатом суммарно.

Итоговая асимптотика решения: $O(N^2 \log N)$.

Задача E. LinearMapReduce

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

В вычислительной парадигме linearMapReduce элементарной единицей данных являются записи. Каждая запись состоит из двух полей — ключа и значения. Ключ и значение в linearMapReduce являются целыми числами из диапазона $[0, 10^9 + 6]$. Для обработки данных в парадигме linearMapReduce используются операторы двух типов — linearMap и гораздо менее популярный linearReduce. Оператор типа linearMap преобразовывает поданную на вход пару ключ-значение ($Key_{in}, Value_{in}$) в другую пару ключ-значение ($Key_{out}, Value_{out}$) по следующему правилу:

$$Key_{out} = (a_{11} \cdot Key_{in} + a_{12} \cdot Value_{in}) \bmod (10^9 + 7)$$

$$Value_{out} = (a_{21} \cdot Key_{in} + a_{22} \cdot Value_{in}) \bmod (10^9 + 7)$$

Как можно видеть, оператор типа linearMap полностью определяется 4-мя целыми числами: $a_{11}, a_{12}, a_{21}, a_{22}$.

Ксении достался в поддержку проект X, в котором активно используется вычислительная парадигма linearMapReduce. Во время очередного масштабирования, Ксении потребовалось эффективно решить следующую задачу:

N операторов типа linearMap M_1, M_2, \dots, M_N заданы своими коэффициентами. Приходят запросы, состоящие из четырех чисел: $Key, Value, a, b$ — пара ключ-значение и два индекса в последовательности операторов. В ответ на запрос необходимо вывести результат применения к записи $(Key, Value)$, операторов с индексами начиная с a -го и заканчивая b -ым.

Обратите внимание, что в случае $a \leq b$ необходимо применить операторы в следующем порядке: $M_a, M_{a+1}, \dots, M_{b-1}, M_b$, а в случае $a > b$ необходимо применить операторы порядке $M_a, M_{a-1}, \dots, M_{b+1}, M_b$.

Формат входного файла

В первой строке задано число $1 \leq N \leq 100000$ — количество операторов типа linearMap. В следующих N строках, по одному в строке, описаны операторы типа linearMap. Каждый оператор задан четверкой чисел перечисленных через пробел: $a_{11}, a_{12}, a_{21}, a_{22}$, где $0 \leq a_{11}, a_{12}, a_{21}, a_{22} \leq 10^9 + 6$. В $N + 2$ -ой строке записано одно число Q ($1 \leq Q \leq 100000$) — количество запросов. Каждый запрос состоит из 4-х чисел: $Key, Value, a, b$ записанных через пробел, где $0 \leq Key, Value \leq 10^9 + 6$ и $1 \leq a, b \leq N$.

Формат выходного файла

Для каждого запроса необходимо вывести два числа — пару ключ-значение получившуюся в результате применения операторов описанных в запросе.

Примеры

stdin	stdout
3 2	15 15
1 0 0 1	20 20
1 2 2 1	
1 3 3 1	
5 5 1 2	
5 5 3 3	

Разбор задачи E. LinearMapReduce

Представленная в условии задачи операция `linearMap` соответствует умножению вектора на матрицу:

$$\begin{pmatrix} Key_{out} \\ Value_{out} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} Key_{in} \\ Value_{in} \end{pmatrix} \quad (1)$$

Учитывая то, что матрицы множатся на вектор слева, для обработки запроса с $a \leq b$ (применения операторов $M_a, M_{a+1}, \dots, M_{b-1}, M_b$) необходимо умножить вектор на матрицу: $M_b \cdot M_{b-1} \cdot \dots \cdot M_{a+1} \cdot M_a$. Чтобы обрабатывать запросы быстро — необходимо построить дерево отрезков, которое в вершине держит произведение матриц в обратном порядке. Для случая $a > b$ необходимо организовать дерево отрезков, которое в вершине держит произведение матриц в прямом порядке.

Асимптотика решения: $O(N + Q \cdot \log N)$.

Задача F. Круги и деревья

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

Задано корневое ориентированное дерево, состоящее из N вершин, пронумерованных числами от 1 до N . Необходимо расположить N окружностей x_i, y_i, r_i на плоскости таким образом, что:

1. Каждое из чисел x_i, y_i, r_i — целое, $-10^6 \leq x_i, y_i \leq 10^6$, $1 \leq r_i \leq 2 \cdot 10^6$;
2. Окружности не имеют общих точек;
3. Окружность i вкладывается в окружность j тогда и только тогда, когда из вершины j заданного корневого дерева можно попасть в вершину i .

Формат входного файла

В первой строке задано число N ($1 \leq N \leq 20000$) — количество вершин дерева. В следующих N строках идет описание корневого ориентированного дерева. В строке i содержится ровно одно число p_i — предок вершины $i - 1$, если $i - 1$ не корень и -1 в противном случае.

Формат выходного файла

В первой строке следует вывести число N . В следующих N строках необходимо вывести окружности, удовлетворяющие условиям 1-3 в формате “ $x_i \ y_i \ r_i$ ” (кавычки выводить, естественно, не нужно).

Примеры

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

Разбор задачи F. Круги и деревья

Построим скобочную последовательность, которая представляет заданное корневое дерево. Найдем для каждой открывающей скобки ее закрывающую. Если их индексы есть l_i и r_i соответственно, то поставим окружность с центром в точке $(l_i + r_i, 0)$ радиуса $r_i - l_i$.

Асимптотика решения: $O(N)$.

Задача G. Петя и игра

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

Маленький Петя очень любит играть в компьютерные игры. Недавно мама подарила ему игру, в которой нужно управлять королевством. В королевстве есть N городов, пронумерованных числами от 1 до N , причем столица имеет номер 1. Некоторые пары городов соединены двухсторонними дорогами, для которых известна их длина в километрах. Всего есть $N - 1$ дорога, причем гарантируется, что от любого города можно добраться до столицы единственным образом, не проходя по одной дороге больше одного раза. В каждом городе королевства, кроме столицы, расположено ровно по одному гонцу. Гонцы характеризуются своей скоростью и временем, которое они тратят на сборы в дорогу. У разных гонцов эти характеристики могут отличаться. Когда на город нападают, оттуда отправляется гонец в столицу. Он тратит необходимое ему время на сборы в дорогу, после чего перемещается по дороге по направлению к столице. Затем, если он еще не доехал до столицы, он может либо

продолжить движение к столице (уже не тратя времени на сборы), либо передать сообщение гонцу, расположенному в том городе, где он находится. В таком случае, все начинается сначала: гонец в начале тратит время на сборы, а затем выдвигается в столицу, делая все то же самое. Таким образом, в доставке сообщения в столицу может принимать участие любое количество гонцов.

Помогите Пете определить для каждого города наименьшее время, за которое сообщение может быть доставлено из него в столицу.

Формат входного файла

Первая строка содержит одно число N ($1 \leq N \leq 100\,000$). Каждая из следующих $N - 1$ строк содержит по 3 целых числа u , v и d ($1 \leq u \leq N$, $1 \leq v \leq N$, $1 \leq d \leq 10\,000$) — номера городов, соединенных дорогой и длину дороги в километрах. Затем идет еще $N - 1$ строка. i -я из них содержит 2 числа S_i и V_i ($0 \leq S_i \leq 10^9$, $1 \leq V_i \leq 10^9$), характеризующих гонца из города $i + 1$. S_i — это время в минутах, необходимое гонцу для сборов, а V_i — это время в минутах, которое гонец тратит на преодоление 1 километра пути.

Формат выходного файла

Выведите $N - 1$ число через пробел, i -е из которых должно равняться наименьшему времени в минутах, которое необходимо для доставки сообщения из города $i + 1$ в столицу.

Примеры

stdin	stdout
5	206 321 542 328
1 2 20	
2 3 12	
2 4 1	
4 5 3	
26 9	
1 10	
500 2	
2 30	

Разбор задачи G. Петя и игра

Запустим обход дерева “в глубину” из корня (вершины номер 1). Пусть в данный момент мы находимся в вершине v . Обозначим через P_v путь из корня в вершину v . Соответственно, j -я вершина, считая от корня, в этом пути

обозначается как $P_v(j)$. Этот путь мы можем выписать в массив прямо в процессе обхода дерева: при попадании в вершину записываем ее в конец списка, а при выходе из нее — удаляем последний элемент из списка. Таким образом, в списке всегда будет корректный путь из корня в текущую вершину. Заметим, что когда мы зафиксировали вершину v , задача становится эквивалентной первой задаче из лекции. В данном случае, координатой X вершины v (обозначается $X(V)$) является сумма весов ребер на пути от корня до v . Итак, пусть $f(v)$ — это минимальное время, требуемое для отправки сообщения из v в корень. Ясно, что $f(1) = 1$. Обозначим глубину вершины v как $h(v)$. Для пересчета $f(v)$ воспользуемся следующей формулой:

$$f(v) = \min_{1 \leq j \leq h(v)} \{ (X(v) - X(P_v(j))) \cdot V_v + S_v + f(P_v(j)) \}$$

По определению функцию $f(v)$ можно посчитать для всех вершин в сумме за $O(N^2)$. Попробуем воспользоваться тем же приемом, который был в лекции. Поставим каждой вершине в соответствие прямую $y = k_v x + l$, где $k = -X(v)$, а $l = f(v)$. Будем хранить нижнее огибающее множество прямых и при добавлении новой прямой перестраивать его. Проблема заключается в том, что при добавлении прямой мы можем удалить часть прямых из нижнего огибающего множества, а при выходе из рекурсии нам придется откатить все изменения, т.е. добавить их обратно. Выходит, что одни и те же прямые могут добавляться в множество и удаляться из него много раз, тем самым не улучшая асимптотику решения, а оставляя ее $O(N^2)$, какой она была раньше.

Для того, чтобы быстро вычислять $f(v)$, нам надо научиться при добавлении новой прямой быстро удалять ненужные прямые из множества, а затем быстро откатывать изменения обратно. Пусть для предыдущих прямых на пути мы это уже сделали. Теперь при помощи бинарного поиска мы можем пересечь нижнее огибающее множество с вертикальной прямой $x = -V_v$ и, таким образом, вычислить значение $f(v)$. Далее вспомним, что каждый раз у нас некоторые прямые удаляются из конца массива, а затем в конец кладется новая прямая. Мы можем бинарным поиском найти количество прямых, которые нужно удалить при добавлении очередной прямой. Теперь мы должны поставить в позицию, найденную бинарным поиском, новую прямую, соответствующую вершине v , затерев при этом то, что лежало на той позиции в массиве раньше, а затем изменить размер массива. Но в начале, запишем старый размер массива, номер позиции, куда мы поставили новую прямую, а также то, что лежало в той ячейке массива раньше. Теперь у нас есть корректное нижнее огибающее множество, а после указателя на конец массива лежит “мусор” в виде прямых, которые раньше принадлежали огибающему множеству, а теперь мы про них забыли. Затем сделаем необходимые для поиска в глубину рекурсивные вызовы от детей вершины. Предположим, что после всех этих вызовов наш массив с нижним огибающим множеством вернулся в

то состояние, которое у него было (с добавленной прямой, соответствующей вершине v , а в конце массива лежит “мусор”). Все что нам нужно сделать — это вернуть массив в то состояние, которое у него было до захода поиска в глубину в вершину v . Для этого мы просто сдвинем указатель на конец массива на старое место, а в ту позицию, где лежала прямая, соответствующая вершине v , запишем старое значение, которое мы сохранили. Итак, наш “мусор” снова превратился в часть корректного нижнего огибающего множества прямых. В итоге, мы научились за $O(\log N)$ добавлять прямую в множество и за $O(1)$ возвращать множество в прежнее состояние. Итоговая сложность решения $O(N \log N)$.

Задача Н. Петя и массивы

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

Маленький Петя очень любит массивы целых чисел. Больше, чем массивы, он любит только играть с маленькой Машей. Недавно Маша получила от мамы в подарок 2 числа N и P . Чтобы проверить их дружбу с Петей, Маша попросила его найти количество массивов, состоящих из N целых чисел и обладающих следующими свойствами:

- Каждый элемент массива — это целое число от 1 до P , включительно.
- Не существует последовательного подмассива ненулевой длины, у которого сумма элементов делится на P .

Помогите Пете найти количество таких массивов. Это число может быть очень велико, поэтому найдите его по модулю $1\,000\,000\,007$ ($10^9 + 7$), чтобы не испугать детей.

Формат входного файла

В первой строке записано T ($1 \leq T \leq 2500$) — количество тестов. В каждой из следующих T строк записаны 2 числа N и P ($1 \leq N \leq 50$, $1 \leq P \leq 50$).

Формат выходного файла

Для каждого теста выведите одно число — искомое количество массивов по модулю $10^9 + 7$. Ответы для разных тестов разделяйте переводами строк.

Примеры

stdin	stdout
3	0
5 4	2
2 3	6
2 4	

Разбор задачи Н. Петя и массивы

Пусть массив A удовлетворяет условиям задачи. Построим для него массив частичных сумм B такой, что $B_i = (A_1 + A_2 + \dots + A_i) \bmod P$. Ясно, что по массиву A массив B находится однозначно. Верно и обратное: мы можем однозначно восстановить массив A зная массив B . Следовательно, между множеством массивов A и множеством массивов B есть биекция, т.е. мощности этих множеств одинаковы. Будем считать количество массивов B . Для этого посмотрим, чему эквивалентны условия массива A в терминах массива частичных сумм. Ясно, что никакие 2 элемента массива B не могут быть одинаковыми, поскольку в противном случае, сумма чисел на отрезке между ними (не включая левый, но включая правый) была бы равна 0 по модулю P . Также понятно, что никакой элемент массива B не может быть равным 0, поскольку тогда сумма на соответствующем префиксе массива A была бы равна 0. Эти 2 условия эквивалентны условию, что в массиве A нет последовательного отрезка чисел с суммой 0 по модулю P . Следовательно, нас интересуют все массивы, состоящие из различных целых чисел от 1 до P . Количество таких массивов равно $C_{P-1}^N \cdot N!$, что и является ответом на задачу.

Задача I. Петя и массив 2

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

Маленький Петя очень любит массивы целых чисел. Недавно мама подарила ему массив, состоящий из N целых чисел. Петя решил найти в нем такой отрезок последовательных чисел, для которого значение выражения $A \cdot S + B$ — максимально. Здесь A и B заданы наперед, а S — это сумма чисел в выбранном отрезке. Обратите внимание, что выбранный отрезок должен иметь ненулевую длину.

Формат входного файла

Первая строка содержит 3 числа N ($1 \leq N \leq 10^6$), A и B ($-10^6 \leq A \leq 10^6$, $-10^6 \leq B \leq 10^6$).

Формат выходного файла

Выведите одно число — искомое максимальное значение выражения $A \cdot S + B$.

Примеры

stdin	stdout
4 2 3 0 -1 -1 0	3
5 3 3 5 -1 4 5 2	48

Разбор задачи I. Петя и массив 2

Заметим, что число b никак не влияет на выбор отрезка, поэтому решим задачу для $b = 0$, а затем прибавим b к ответу. Домножим все числа в массиве на a . Теперь задача свелась к поиску отрезка последовательных чисел в массиве с максимальной суммой. Есть несколько способов ее решения за $O(N)$. Самым простым для объяснения является такой. Для нашего массива A построим массив частичных сумм B , такой что $B_i = A_1 + A_2 + \dots + A_i$. Будем считать, что $B_0 = 0$. Тогда сумма чисел на отрезке от i до j равна $B_j - B_{i-1}$. Переберем правый конец отрезка (i) и найдем для него левый конец (j), который максимизирует сумму на отрезке. Ясно, что в качестве левого конца нужно выбрать такое j ($j \leq i$), для которого B_{j-1} минимально. Для этого, будем просто двигать i слева направо и поддерживать в отдельной переменной минимальное значение B_i для всех пройденных элементов. Сложность решения $O(N)$.

Для этой задачи также существует решение за $O(N)$, которое требует $O(1)$ дополнительной памяти. Поиск этого решения оставим в качестве упражнения.

Задача J. Петя и прямоугольники

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

Маленький Петя очень любит прямоугольники. Петя дал маме список прямоугольников, которые он хочет получить в подарок на Новый Год. Каждый прямоугольник характеризуется w и высотой h . Мама хочет сделать Пете приятное и купить все прямоугольники из его списка. Мама отправилась в магазин и узнала, что цена одного прямоугольника равна его площади. К ее счастью, в магазине действует предновогодняя акция, позволяющая покупать прямоугольники не по одному, а сразу наборами. Стоимость одного набора равна ширине самого широкого прямоугольника, умноженной на высоту самого высокого прямоугольника из этого набора. Обратите внимание, что поворачивать прямоугольники (тем самым меняя местами ширину и высоту) нельзя. Помогите маме Пети купить все прямоугольники из списка ее сына, потратив на это наименьшее количество денег.

Формат входного файла

В первой строке записано число N ($1 \leq N \leq 200\,000$) — количество прямоугольников в списке Пети. В каждой из следующих N строк записаны по 2 целых положительных числа, не превышающих 10^6 — ширина и высота очередного прямоугольника.

Формат выходного файла

Выведите одно число — наименьшее количество денег, которое может потратить мама чтобы купить Пете все прямоугольники из его списка.

Примеры

stdin	stdout
4 100 1 15 15 20 5 1 100	500
5 1 10 2 20 3 30 4 40 10 1	170

Разбор задачи J. Петя и прямоугольники

Для начала заметим, что если прямоугольник (w_1, h_1) вкладывается в прямоугольник (w_2, h_2) (выполняется $w_1 \leq w_2$ и $h_1 \leq h_2$), то первый прямоугольник можно удалить из набора без изменения ответа на задачу. Действительно, если оба этих прямоугольника будут находиться в одном наборе при покупке, тогда первый прямоугольник достанется нам “бесплатно”, поскольку он будет полностью покрыт вторым. Значит, мы можем удалить из рассмотрения прямоугольник (w_1, h_1) , а затем при покупке включить его в тот же набор, в котором находился (w_2, h_2) . Следовательно, мы можем оставить в наборе только попарно не вкладывающиеся прямоугольники, а остальные удалить из рассмотрения. Это можно сделать при помощи сортировки и линейного прохода по массиву, суммарно за $O(N \log N)$.

Итак, после того как мы удалили вкладывающиеся прямоугольники, отсортируем все оставшиеся по возрастанию w_i . Легко видеть, что при возрастании i у прямоугольников будет убывать h_i . Действительно, если бы существовало такое i , что $h_{i+1} \leq h_i$, то прямоугольник i вкладывался бы в $i + 1$, а, значит, был бы удален из рассмотрения.

Сделаем еще одно наблюдение. Существует оптимальный способ покупки прямоугольников, при котором в одном наборе всегда будут лежать прямоугольники, идущие подряд в отсортированном списке по w_i . Действительно, стоимость набора прямоугольников равна $w_i \times h_j$, где i — индекс прямоугольника с наименьшим w (следовательно, с наибольшим h), а j — индекс прямоугольника с наибольшим w . Значит, все прямоугольники между i и j нам достанутся “бесплатно”, то есть мы можем их включить в тот же набор, в котором находятся i и j без ухудшения ответа.

Теперь сделанные наблюдения позволяют нам решать задачу при помощи динамического программирования. Обозначим через $f(i)$ минимальное количество денег, которое необходимо потратить чтобы купить все прямоугольники с номерами от 1 до i , включительно. Пусть $f(0) = 0$. Тогда можем записать:

$$f(i) = \min_{1 \leq j \leq i} \{w_i \cdot h_j + f(j - 1)\}$$

Подсчет этой динамики напрямую по приведенной формуле занимает $O(N^2)$ операций. Однако, если считать минимум подходом, описанным в лекции, то можно добиться асимптотики $O(N)$ после сортировки. В данном случае, в уравнении прямой мы возьмем $k = h_j$ и $l = f(j - 1)$. Итоговая сложность решения $O(N \log N)$.

Задача K. Игровелоперы

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	512 Мб

Ксения и Александра работают старшими игровелоперами¹ в одной известной аутсорсинговой компании. Они разработали очередную игру под названием ЛПРКБД². Сейчас они хотят сыграть в последний раз, прежде чем отослать игру заказчикам.

Доска для ЛПРКБД состоит из одного полного, сбалансированного корневого бинарного дерева состоящего из $N = 2^h - 1$ вершин. В вершинах этого дерева могут располагаться бонусы, но не более одного в каждой вершине. В добавок к доске, в комплекте содержится бесконечно много белых и черных фишек. Правила изобретенные Ксенией и Александрой достаточно просты: изначально на доске расположена только одна фишка белого цвета в некоторой вершине V_{start} дерева. Два игрока по очереди ходят. Во время своего хода игрок может выполнить один из трех ходов:

1. “Пойти влево” — передвинуть все фишки (вне зависимости от их цвета) в левого потомка вершин в которых они расположены.
2. “Пойти вправо” — передвинуть все фишки (вне зависимости от их цвета) в правого потомка вершин в которых они расположены.
3. Ход “Разделится” содержит три подшага:
 - Заменить каждую из оставшихся фишек на столе (вне зависимости от ее цвета) на одну белую и одну черную фишку.

- Применить ход “Пойти влево” к белым фишкам и только к ним.
- Применить ход “Пойти вправо” к черным фишкам и только к ним.

Достигнув листа дерева, фишка снимается с доски. В конце концов, когда фишек не остается на доске, игроки подводят итоги. Они подсчитывают количество бонусов в вершинах посещенных хотя бы одной из фишек во время игры. Если их количество оказывается четным — выигрывает Ксения, в противном случае — Александра. Задано полное сбалансированное корневое бинарное дерево и расположение всех бонусов на нем, определите победителя считая, что Ксения и Александра играют оптимально. Ксения ходит первой.

Формат входного файла

Первая строка содержит целое число $N = 2^h - 1$ ($1 \leq N \leq 32\,767$), количество вершин дерева. Следующие N строк описывают полное сбалансированное дерево использованное для игры в ЛПРКБД. В i -ой из этих строк находятся три числа: $left$, $right$ и $bonus$ где:

- $left$ номер левого сына i -ой вершины или 0 если левого сына у нее нет.
- $right$ номер правого сына i -ой вершины или 0 если правого сына у нее нет.
- $bonus$ равен 1 если i -ая вершина содержит бонус и 0 в противном случае.

Формат выходного файла

В i -ой строке должно быть выведено имя победителя (либо “Ksusha”, либо “Sasha”, без кавычек), в предположении, что стартовая позиция находилась в i -ой вершине (разыгрывая игру с $V_{start} = i$), и Ксения ходила первой.

Примеры

stdin	stdout
3	Ksusha
2 3 0	Sasha
0 0 1	Ksusha
0 0 0	

Примечание

¹ - практически настоящая должность.

² - игра в Лево-Право-Разделить на Корневом Бинарном Дереве.

Разбор задачи К. Игровелоперы

Рассмотрим задачу без возможности использовать операцию “Разделиться”. Тогда задача становится классическим упражнением на подсчет выигрышности/проигрышности позиций в игре. Чтобы добавить убранный ход необходимо заметить, что когда игрок его разыгрывает, он на самом деле идет в неявное третье поддерево в вершинах которого находятся бонусы равные скору бонусов соответствующих вершин из левого и правого поддерева. Таким образом, асимптотическую сложность решения можно оценить как $T(h) = 3 \cdot T(h - 1) + 1 = O(3^h)$.

Задача L. Путешествие (Высшая лига)

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

Маленький Петя очень любит путешествовать. В стране Берляндия, где он живет, есть N городов, расположенных на одной прямой. Петя пронумеровал их числами от 1 до N в порядке увеличения красоты. Петя находится в городе 1 и хочет попасть в город N . Чтобы не портить впечатления о поездке, он может посещать города только в порядке увеличения номеров (а, следовательно, и красоты). Для перемещения между городами Петя решил воспользоваться услугами единственной авиакомпании страны — Berland Airlines. Стоимость перелета из города i в город j равна $c_i \cdot |x_i - x_j| + t_j$, где x_i — координата города i , x_j — координата города j , c_i — стоимость единицы самолетного топлива в городе i , а t_j — стоимость въезда в город j . Чтобы было о чем рассказать друзьям, Петя хочет потратить как можно больше (да-да, именно больше) денег на эту поездку. Помогите ему в этом. Обратите внимание, что Пете не обязательно бывать во всех городах.

Формат входного файла

Первая строка содержит целое число N — количество городов в Берляндии ($1 \leq N \leq 100\,000$). Далее следуют N строк. Стока с номером i из них содержит 3 целых числа — x_i ($-10^6 \leq x_i \leq 10^6$), c_i ($1 \leq c_i \leq 10^6$) и t_i ($1 \leq t_i \leq 10^6$). Все координаты x_i различны.

Формат выходного файла

Выведите искомое наибольшее количество денег, которые Петя может потратить чтобы добраться из города 1 в город N . Гарантируется, что ответ не превышает 10^{12} .

Примеры

stdin	stdout
4	123
5 10 2	
0 1 10	
15 3 14	
17 2 3	
1	0
709 50 8	

Примечание

Два возможных оптимальных маршрута для первого теста: $1 \rightarrow 4$, $1 \rightarrow 3 \rightarrow 4$.

Разбор задачи L. Путешествие (Высшая лига)

Будем решать задачу при помощи динамического программирования. Обозначим через $f(i)$ наибольшую стоимость поездки из города 1 в город i . Ясно, что $f(1) = 1$. Для остальных городов f вычисляется по следующей формуле:

$$f(i) = \max_{1 \leq j \leq i} \{|x_i - x_j| \cdot c_j + t_i + f(j)\}$$

Раскроем модуль, рассмотрев при этом 2 случая. Сразу вынесем все, что не зависит от j из под максимума.

1. Если $x_i < x_j$, то

$$f(i) = \max_{1 \leq j \leq i} \{x_i \cdot c_j + f(j) - x_j \cdot c_j\} + t_i$$

2. Если $x_i \geq x_j$, то

$$f(i) = \max_{1 \leq j \leq i} \{-x_i \cdot c_j + f(j) + x_j \cdot c_j\} + t_i$$

Сделаем важное наблюдение. Если мы раскроем модуль неправильно, то значение $f(i)$ станет только меньше, поэтому мы можем попробовать раскрыть модуль обоими способами и выбрать из них максимум — он и будет правильным. Опять же, рассмотрим прием из лекции для поиска максимума. Для каждого города добавим в верхнее огибающее множество не одну, а 2 прямые вида $y = kx + l$. У одной $k = c_j$, а $l = f(j) - x_j \cdot c_j$, а у другой $k = -c_j$, а

$l = f(j) + x_j \cdot c_j$. Есть еще одна проблема: в отличии от лекционных задач, здесь города не отсортированы по x_i , а значит, добавлять прямые в конец массива нельзя — нам может понадобиться добавить прямую куда-то в середину, при этом удалив несколько прямых опять же из середины массива.

Для того, чтобы как и раньше, быстро поддерживать верхнее огибающее множество, будем хранить все прямые в дереве. Это может быть любое сбалансированное двоичное дерево (например, декартово), дерево отрезков или даже `std::set` (в C++). Прямые в дереве будут отсортированы по возрастанию углового коэффициента k . Тогда, при добавлении новой прямой, мы сперва найдем ее позицию в дереве (считайте, сделаем `lower_bound` в массиве, отсортированном по k), после чего удалим лишние прямые непосредственно справа и слева от нее. Проверка делается аналогично тому случаю, когда мы добавляем прямую в конец массива. Следует обратить внимание, что еще нужно отдельно проверить, нужно ли вообще вставлять нашу прямую в множество, т.к. возможен случай, когда этого делать не требуется. Пересекать верхнее огибающее множество с вертикальной прямой будем точно также, как и в случае с массивом, только теперь вместо бинарного поиска у нас будет спуск по дереву. Итоговая сложность решения $O(N \log N)$.

Задача М. Петя и среднее (Высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Маленький Петя очень любит математику. Недавно учительница задала детям в качестве домашнего задания следующую задачу. Даны 2 массива A и W , состоящие из N целых чисел. Требуется найти в массиве A отрезок подряд идущих чисел с максимальным средним взвешенным значением с весами из массива W (см. определение снизу). Петя решил эту задачу очень просто: он доказал, что ответ достаточно искать только среди отрезков длины 1, а затем перебрал все такие отрезки и выбрал лучший. Петя захотел усложнить самому себе задачу, сказав что он будет искать отрезок длины не меньшей L . Помогите ему решить эту новую, усложненную задачу.

Формат входного файла

Первая строка содержит 2 целых числа N и L ($1 \leq L \leq N \leq 10^6$) — количество чисел в массиве и наименьшую длину искомого отрезка. Следующая строка содержит N целых чисел по модулю не превосходящих 10^3 — массив

A. Следующая строка содержит N целых положительных чисел не превосходящих $2 \cdot 10^9$ — массив W .

Формат выходного файла

Выведите 2 числа — индексы начала и конца отрезка с максимальным средним взвешенным значением. Длина найденного отрезка должна быть не меньше L . Элементы массива нумеруются начиная с единицы. Если оптимальных отрезков несколько — выведите любой.

Примеры

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

Примечание

В данной задаче под средним взвешенным значением чисел из массива A с весами W на отрезке от i до j ($i \leq j$) понимается следующая величина:

$$\frac{A_i \cdot W_i + A_{i+1} \cdot W_{i+1} + \dots + A_j \cdot W_j}{W_i + W_{i+1} + \dots + W_j}$$

Разбор задачи М. Петя и среднее (Высшая лига)

Неправильное решение Данную задачу очень хочется решать следующим образом. Сделаем бинарный поиск по ответу. Пусть мы хотим проверить, существует ли отрезок $[l, r]$ со средним взвешенным значением не меньшим X . Для этого, запишем формулу:

$$\frac{A_l \cdot W_l + A_{l+1} \cdot W_{l+1} + \dots + A_r \cdot W_r}{W_l + W_{l+1} + \dots + W_r} \geq X$$

После тождественных преобразований получим выражение:

$$W_l \cdot (A_l - X) + W_{l+1} \cdot (A_{l+1} - X) + \dots + W_r \cdot (A_r - X) \geq 0$$

Введем обозначение $B_i = W_i \cdot (A_i - X)$. Теперь нам нужно проверить, существует ли отрезок с неотрицательной суммой. Для этого, найдем отрезок с

максимальной суммой и сравним ее с нулем. Алгоритм нахождения отрезка с максимальной суммой можно найти в задаче I этого контеста. Сложность решения $O(N \log MAX)$, где MAX — это максимальное значение ответа. Проблема заключается в том, что такое решение находит ответ в `double`, с какой-то заданной точностью. В этой задаче есть тесты, где существуют 2 отрезка со средним взвешенным значением отличающимся на величину порядка 10^{-36} . Следовательно, никакой встроенный тип данных не сможет отличить такие отрезки, поэтому приведенное выше решение нам не подходит.

Правильное решение Будем перебирать правый конец отрезка слева направо. Пусть сейчас у нас зафиксирован правый конец в i . Пусть левый конец находится на позиции j . Тогда взвешенное среднее значение X находится по следующей формуле:

$$X = \frac{A_j \cdot W_j + A_{j+1} \cdot W_{j+1} + \cdots + A_i \cdot W_i}{W_j + W_{j+1} + \cdots + W_i}$$

Пусть $S_i = A_1 \cdot W_1 + A_2 \cdot W_2 + \cdots + A_i \cdot W_i$, а $T_i = W_1 + W_2 + \cdots + W_i$. Пусть $S_0 = 0$ и $T_0 = 0$. Перепишем в терминах новых двух массивов формулу, записанную выше:

$$X = \frac{S_i - S_{j-1}}{T_i - T_{j-1}}$$

Теперь домножим все на знаменатель и перенесем в одну сторону то, что зависит только от i , а в другую — только от j .

$$S_i - T_i \cdot X = S_j - T_j \cdot X$$

Поставим в соответствие каждому i прямую $y = k_i \cdot x + l_i$, где $k_i = -T_i$, а $l_i = S_i$. Построим для всех j ($j < i$) нижнее огибающее множество соответствующих им прямых. Утверждается, что для данного фиксированного i (правого конца) оптимальное j (левый конец) можно найти следующим образом: пересечем прямую $y = k_i \cdot x + l_i$ с уже построенным нижним огибающим множеством, запомнив при этом, с частью какой прямой, входящей в нижнее огибающее множество, пересеклась наша прямая для элемента i . Номер j той прямой и будет оптимальным левым концом. При этом утверждается, что у нашей прямой и нижнего огибающего множества всегда будет ровно одна точка пересечения. Последнее доказать легко, поскольку $W_i > 0$, а значит, T_i монотонно возрастает, т.е. угловой коэффициент k_i убывает при увеличении i . То, что такая прямая j будет оптимальной, доказать тоже не сложно. Сделаем это от противного. Пусть оптимальная прямая не принадлежит нижнему огибающему множеству. Тогда будет существовать прямая, принадлежащая нижнему огибающему множеству, которая в координате X точки пересечения

и нашей прямой $y = k_i \cdot x + l_i$ будет иметь меньший Y . Поскольку прямая $y = k_i \cdot x + l_i$ имеет строго меньший угловой коэффициент, чем все прямые из нижнего огибающего множества, она пересечется с огибающим множеством где-то правее, а значит, предположение об оптимальности ответа было неверным.

Пересекать нижнее огибающее множество с произвольной прямой можно точно также, как и с вертикальной. Мало того, в данной задаче можно поддерживать указатель на текущую прямую, дающую оптимальную точку пересечения и двигать его только вправо, поскольку нас интересует нахождения максимального X среди всех таких точек пересечения. Итоговая сложность решения $O(N)$.

Последнее, что следует сказать, это то, что при промежуточных вычислениях нужно быть предельно аккуратными, поскольку числитель и знаменатель ответа может быть очень большим и сравнивать дроби нужно так, чтобы избежать переполнение long long.

Задача N. Депутаты на дереве (Высшая лига)

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

В следствии чересмерной децентрализации, в стране X исчезла столица. Теперь в стране X, сложная политическая ситуация, урегулировать которую можно лишь собрав новый совет депутатов.

В каждом из N городов страны X живут депутаты. В городе i ($1 \leq i \leq N$) живет ровно d_i депутатов. Между городами страны X проложено $N - 1$ двусторонних дорог, по которым можно ровно одним путем добраться из каждого города в каждый другой. Для каждой дороги i ($1 \leq i \leq N - 1$) известны города, которые она соединяет u_i и v_i , а также стоимость проезда одного человека w_i по этой дороге.

Так как с финансами у страны X существуют серьезные проблемы, для спасения страны необходимо собрать совет депутатов в каком-то городе таким образом, чтобы суммарные затраты всех депутатов были наименьшими. Для упрощения задачи, вам как и.о. премьер-министра страны X разрешается объявить бесплатным проезд на $0 \leq K \leq N - 1$ дорогах.

Формат входного файла

В первой строке указаны два числа N и K , ($1 \leq N \leq 100000$, $0 \leq K \leq N - 1$). Во второй строке расположено N целых чисел от 1 до

100000 — количество депутатов проживающих в i -ом городе. В оставшихся $N - 1$ строке описаны дороги в стране X. Каждая дорога задана в отдельной строке тремя числами: u_i, v_i и w_i ($1 \leq u_i, v_i \leq N, 1 \leq w_i \leq 100000$). Гарантируется, что из каждого города по заданным дорогам можно добраться в любой другой.

Формат выходного файла

Выведите единственное число — минимальное количество денег, которое придется потратить депутатам, чтобы собраться на совет в один город.

Примеры

stdin	stdout
4 1 10 10 10 1 1 2 1 1 3 10 3 4 1000	120

Разбор задачи N. Депутаты на дереве (Высшая лига)

В этой задаче нужно было выбрать вершину в дереве в которой дешевле всего собраться депутатам, с учетом того, что на K ребрах можно отменить плату за проезд. Переберем вершину где будут собираться депутаты. Тогда, для каждого депутата ясно по каким ребрам ему предстоит идти в вершину сбора. Подсчитаем для каждого ребра суммарное количество денег которые заплатят депутаты проходя по нему. Зная персональный вклад в суммарный ответ для каждого ребра можно определить K самых весомых ребер и удалить их. Таким образом для каждой выбранной вершины сбора за время $O(N \log N)$ можно посчитать минимальную стоимость которую необходимо уплатить депутатом суммарно. Получаем наивное решение за $O(N^2 \log N)$.

Однако, можно заметить, что при оптимальном расположении вершины сбора, вклад каждого ребра нам известен. Рассмотрим ребро (u_i, v_i) веса w_i . Пускай количество депутатов в дереве со стороны вершины u_i (включая вершину) обозначено как L , а со стороны v_i — как R . Для определенности предположим что $L \leq R$. Тогда верно, что в оптимальном ответе, вне зависимости от того обнуляем ли мы текущее ребро или нет, всегда выгодно чтобы вершина сбора лежала со стороны вершины v_i . И правда, предположив обратное и сравнив получаемую стоимость сбора в некоторой вершине со стороны вершины u_i с той, что можно получить в вершине v_i мы получим противоречие. Таким образом, для каждого ребра нам известно сколько оно вносит в финальный ответ — $\min(L, R) \cdot w_i$. Аналогично, с наивным решением, можно

воспользоваться жадностью — отсортировать ребра по этой величине и выкинуть K самых дорогих.

Итоговая асимптотика решения: $O(N \log N)$.

День пятый (19.02.2014 г.) Конкурс Романа Андреева

Об авторе...

Андреев Роман, родился 5 октября 1993 г. в городе Троицк Московской области. С 7 по 9 класс учился в московской школе “Интеллектуал”, а с 10 по 11 в СУНЦ МГУ(Специализированный учебно-научный центр Московского государственного университета им. М.В. Ломоносова — школа им. А.Н. Колмогорова).



Основные достижения:

- В 9 классе получил серебряную медаль на РОИ, в 10 классе призер, в 11 классе абсолютное 2 место.
- Серебряная медаль IOI 2010.
- Студент кафедры Статистического моделирования Математико-механического факультета Санкт-Петербургского государственного университета.
- Победитель Facebook Hacker Cup 2012.
- Финалист ACM ICPC World Finals 2012 (18–35 место).
- 4 место на турнире ICL, 5 место на Чемпионате урала в составе команды “Angry Muffin”.
- Финалист Yandex.Algorithm 2013.
- 5 место на NEERC 2013 в составе команды “Теперь Банановый!”.

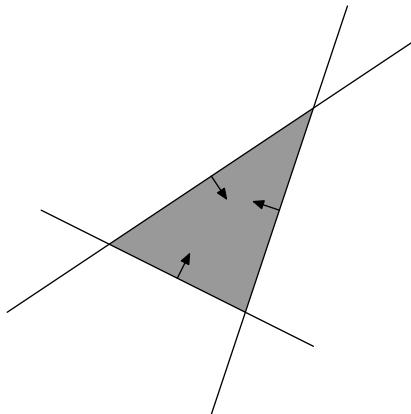
Теоретический материал. Лекция про пересечение полу- плоскостей

Постановка задачи и базовые рассуждения

Рассмотрим классическую задачу — даны n полуплоскостей, нужно найти множество точек, таких что они лежат во всех полуплоскостях.

Лемма 1. *Множество точек, лежащих в нашем пересечении — выпуклое множество.*

Доказательство. По определению множество называется выпуклым, если из того, что две точки лежат в этом множестве, следует, что и отрезок их соединяющий тоже лежит в этом множестве. А если пара точек лежит в полуплоскости, то и отрезок их соединяющий тоже лежит в полуплоскости, значит наше множество — выпуклое. \square



Так как полуплоскости — объект бесконечный, а в компьютере нам не удобно работать с бесконечными объектами, мы введем так называемый *bounding box* — скажем, что мы живем в области с координатами не превосходящими какой-нибудь большой константы (например 10^9), которая обычно зависит от задачи. Заметим, что это квадрат, а это значит, что мы можем задать его, как пересечение четырех полуплоскостей, соответствующих сторонам квадрата.

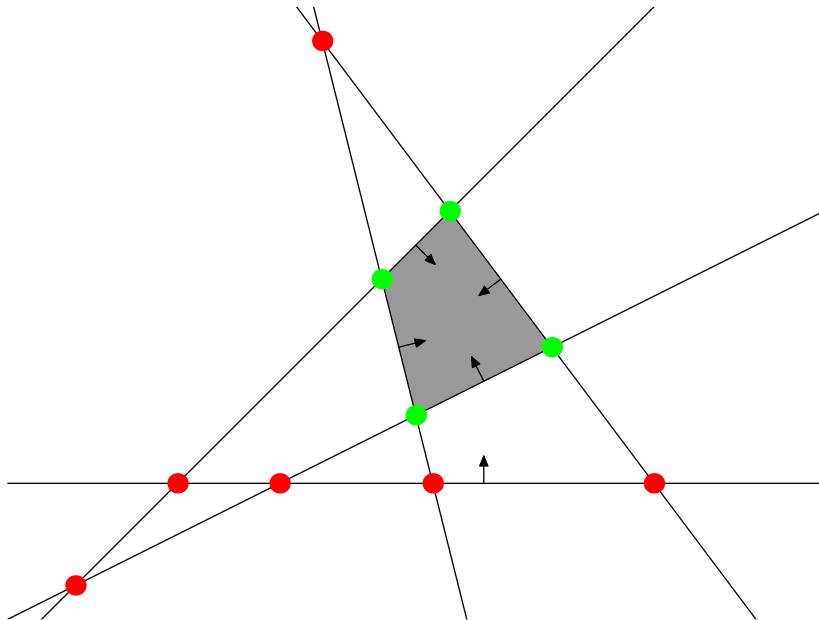
Лемма 2. *Множество точек, лежащих в нашем пересечении — либо выпуклый многоугольник (здесь мы уже используем тем, что мы живем в ограниченной области), либо пустое множество.*

Доказательство. В принципе это утверждение очевидно, но формально его можно доказать индукцией по n . Для нуля полуплоскостей, как мы уже выяснили, область — квадрат. База доказана. Шаг — при сечении выпуклого многоугольника полуплоскостью мы получаем либо многоугольник, либо пустое множество. \square

Алгоритм за $O(n^3)$

Давайте поймем, какие вершины будут являться вершинами нашего многоугольника. Понятно, что вершина — это пересечение двух прямых, соответствующих каким-то двум полуплоскостям. Давайте переберем все пары прямых, которые в пересечении дают точку и запишем все такие точки в список. Точек в списке будет не более $\frac{n(n-1)}{2}$.

Теперь для каждой точки проверим за $O(n)$ лежит ли она во всех полуплоскостях. После этого у нас останутся только точки, являющиеся вершинами ответа, только они будут идти в произвольном порядке. Если нам нужен весь многоугольник в порядке обхода, то нам придется в конце отсортировать их по углу, относительно самой левой точки (на самом деле это первый шаг в алгоритме Грэхема построения выпуклой оболочки). Но чаще всего нам нужно всего лишь проверить не пусто ли пересечение, или выделить одну точку из пересечения. Во втором случае нам удобнее будет взять центр масс вершин, тогда он будет строго лежать внутри всех полуплоскостей, в отличие от граничных точек, которые лежат не строго внутри.

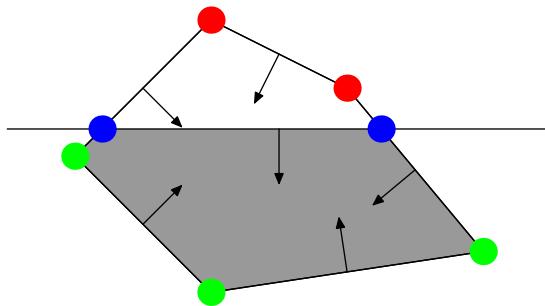


Итоговая асимптотика — $O(n^3)$. Опишем неасимптотическую оптимизацию — понятно, что если точка уже не лежит в какой-то из полуплоскостей, то идти дальше не нужно. Чтобы избавиться от фиксированного порядка мы применим случайную перестановку к полуплоскостям, что обычно дает ускорение примерно в два раза.

Алгоритм за $O(n^2)$

Давайте внимательно посмотрим на доказательство второй леммы. Давай делать ровно тоже самое. На каждом шаге будем поддерживать многоуголь-

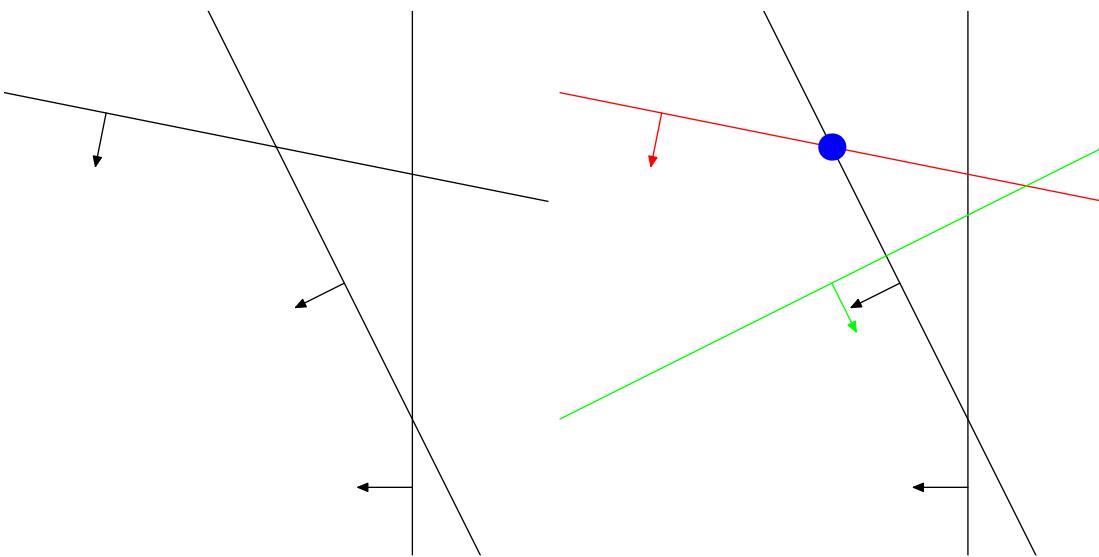
ник, который получается в пересечении первых k полуплоскостей. Будем добавлять полуплоскости по-очереди. Если все вершины текущего многоугольника лежат в $(k+1)$ -й полуплоскости, то мы переходим к следующей. Если все вершины лежат вне, то мы понимаем, что пересечение пусто. Остается случай, когда прямая пересекает наш многоугольник дважды. Заметим, что в ответ войдут те вершины текущего многоугольника, которые лежат в $(k+1)$ -й полуплоскости, а также две точки пересечения. В итоге получаем следующую процедуру: идем по сторонам многоугольника, если обе вершины лежат внутри, то добавляем вторую в ответ, если они по разные стороны, то добавляем точку пересечения в ответ, если они вне, то ничего не добавляем.



В результате мы получим многоугольник пересечения в порядке обхода, причем на каждой итерации мы сделаем $O(k)$ действий, значит суммарно наш алгоритм будет работать за $O(n^2)$. Также заметим, что наш алгоритм *online*, то есть нам не обязательно знать все полуплоскости в начале, мы можем обрабатывать запросы по ходу. Алгоритм за $O(n^3)$ не обладает таким свойством — нам нужно знать все полуплоскости заранее, то есть он *offline*. Заметим, что оптимизация с применением случайной перестановки улучшает и этот алгоритм в два раза.

Алгоритм за $O(n \log n)$

Давайте будем делать что-то очень похожее на алгоритм Грэхема построения выпуклой оболочки — сначала отсортируем полуплоскости, а потом будем идти алгоритмом со стеком. Отсортируем полуплоскости по углу вектору нормали, при равенстве упорядочим по расстоянию от точки $(0, 0)$ (если делать совсем честно, то нужно из всех полуплоскостей с одинаковым вектором нормали оставить одну самую вложенную). Теперь будем добавлять полуплоскости в отсортированном порядке и смотреть, как меняется пересечение. Будем хранить полуплоскости в стеке. При добавлении могут удаляться какие-то старые полуплоскости.



Пока точка пересечения двух последних прямых на стеке будет лежать вне нашей добавляемой полуплоскости, то, как видно из рисунка, мы должны удалить последнюю прямую со стека. После удаления всех лишних прямых мы кладем нашу полуплоскость на стек и продолжаем процесс. Заметим, что в итоге мы получим не совсем то, что мы хотели — самая первая прямая, которую мы положим на стек так никогда из него и не уйдет. То есть в стеке у нас будет сначала какое-то количество стартовых неверных полуплоскостей, затем кусок пересечения, а потом опять в хвосте будут неверные. Для того, чтобы это побороть мы повторим операцию еще раз, то есть снова попробуем добавить все n полуплоскостей в стек. Тогда у нас в начале и в конце будут все те же неверные полуплоскости, но посередине будет несколько раз намотанные на многоугольник ответа верные полуплоскости. Теперь перед нами стоит задача выделения правильного куска из середины. Заметим, что если полуплоскость лежит два раза в стеке, то она входит в ответ, причем ответом будет являться ровно то множество полуплоскостей, которые лежат между двумя вхождениями нашей полуплоскости.

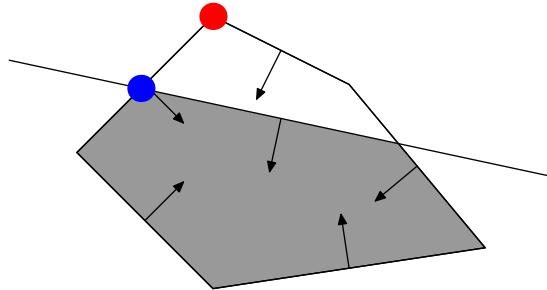
Заметим, что если полуплоскости нам уже даны в отсортированном порядке (например стороны выпуклого многоугольника), то наш алгоритм будет работать за $O(n)$.

Алгоритм за $O(n)$

Алгоритм, который мы приведем далее отличается от вышеописанных тем, что во-первых он будет искать не весь многоугольник пересечения, а только одну точку внутри (или говорить, что пересечение пусто), во-вторых он будет рандомизированный. Аналогично алгоритму за $O(n^2)$ будем добавлять полуплоскости по очереди, поддерживая самую верхнюю точку из многоугольника пересечения. На самом деле, так как мы можем повернуть картину ми-

ра на любой угол, то наш алгоритм будет поддерживать самую удаленную по какому-то фиксированному направлению точку.

Пусть точка p является верхней в пересечении первых k полуплоскостей. Попробуем добавить $(k + 1)$ -ю полуплоскость. Если точка p уже в ней лежит, то она и останется верхней, так как множество точек не может увеличиться. В том случае, если p не лежит в ней, то нужно пересчитать верхнюю точку. Понятно, что либо у нас пересечение стало пусто, либо верхняя точка будет лежать на нашей новой прямой. Тогда каждая полуплоскость в пересечении с $(k + 1)$ -й прямой будет давать луч (либо пустое множество, либо всю прямую, но нас эти случаи пока не интересуют). Лучи бывают двух типов – идущие слева-направо и справа-налево (если смотреть в проекции на нашу новую прямую). Понятно, что пересечением лучей первого типа будет также луч, причем выходящий из самого правого начала, аналогично для второго типа лучей. В итоге нашей новой точкой будет либо самое правое начало лучей идущих слева-направо, либо самое левое начало лучей идущих справа-налево. Если эти обе точки будут лежать вне пересечения первых k полуплоскостей, то пересечение пусто.



Пока что мы получили алгоритм за $O(n^2)$ и абсолютно непонятно, откуда может взяться линейная асимптотика, ведь в худшем случае для каждой прямой нам нужно будет пересчитывать новую верхнюю точку, что также будет происходить за $O(n)$. Давайте в очередной раз применим случайную перестановку к полуплоскостям. Это и даст нам выигрыш в асимптотике. Оценим вероятность того, что на k -м шаге нам нужно будет пересчитывать верхнюю точку. Мы знаем, что эта точка получается в результате пересечения каких-то двух прямых, которые являются соседними в многоугольнике-ответе для $(k + 1)$ -го шага. Если одна из этих прямых попадет на $(k + 1)$ -е место, то нам придется пересчитать верхнюю точку. В любом другом случае получится, что эти две прямые образовывали пересечение еще до $(k + 1)$ -го шага и ничего пересчитывать не нужно. Значит вероятность пересчитывания будет равна $\frac{2}{k+1}$. Причем пересчитываем мы за $O(k)$. Значит суммарная асимптотика будет равна $O\left(\sum_{k=1}^n (k + 1) \frac{2}{k+1}\right) = O(n)$, что и требовалось доказать.

Задачи и разборы

Задача А. Гарри Поттер и три заклинания

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Давным-давно (наверное, ещё в первой книге) великий алхимик, создатель философского камня Николас Фламель научил Гарри Поттера трём полезным заклинаниям. Первое из них позволяет превратить a граммов песка в b граммов свинца, второе — c граммов свинца в d граммов золота, а третье — e граммов золота в f граммов песка. Когда Гарри рассказал об этих заклинаниях своим друзьям, Рон Уизли был в восторге: ведь если получится превращать песок в свинец, свинец в золото, часть золота — снова в песок и так далее, то можно будет, начиная с небольшого количества песка, получить огромное количество золота! Даже бесконечное количество золота! Гермиона Грейндженер, напротив, отнеслась к этой идеи скептически. Она утверждает, что, согласно закону сохранения материи, невозможно получить бесконечное количество материи даже при помощи магии. Наоборот, количество материи при превращениях может даже уменьшаться, переходя в магическую энергию. Несмотря на то, что аргументы Гермионы выглядят убедительно, Рон не собирается ей верить. По его мнению, Гермиона придумала свой закон сохранения материи только для того, чтобы Гарри с Роном перестали заниматься ерундой, а лучше — шли учить уроки. Поэтому Рон уже набрал некоторое количество песка для экспериментов и, кажется, ссоры между друзьями не избежать...

Помогите Гарри определить, кто из его друзей прав, и всё-таки предотвратить ссору. Для этого вам придется выяснить, можно ли из некоторого конечного количества песка получить количество золота, большее любого наперёд заданного числа.

Формат входного файла

В первой строке заданы шесть целых чисел a, b, c, d, e, f ($0 \leq a, b, c, d, e, f \leq 1000$).

Формат выходного файла

Выведите “Ron”, если, имея некоторое конечное количество песка (и не имея вообще золота и свинца), возможно получить сколь угодно большое количество золота, то есть прав Рон. В противном случае выведите “Hermione”.

Примеры

stdin	stdout
100 200 250 150 200 250	Ron
100 50 50 200 200 100	Hermione
100 10 200 20 300 30	Hermione

Примечание

Разберём первый пример. Начнём с 500 граммов песка. Применяя 5 раз первое заклинание, превратим их в 1000 граммов свинца. Затем 4 раза применим второе заклинание и получим 600 граммов золота. Из них выделим 400 и превратим их снова в песок. Получим 500 граммов песка и 200 граммов золота. Применяя все те же операции к 500 граммам песка повторно, можно будет каждый раз получать дополнительные 200 граммов золота. Таким образом можно получить 200, 400, 600, ... граммов золота, то есть, начиная с конечного количества песка (500 граммов), можно получить количество золота, большее любого наперёд заданного числа.

Разбор задачи А. Гарри Поттер и три заклинания

Если $a, b, c, d, e, f > 0$, то нам просто нужно проверить, что $\frac{bdf}{ace} > 1$. Остается аккуратно разобрать случаи, когда какие-то параметры равны 0.

Задача В. Антисортировка

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Обычно в условии задач вам говорят, что необходимо сделать. Нам это кажется скучным. В этой задаче мы поступим наоборот. Скажем, что вы должны не сделать:

Вы не должны отсортировать данную последовательность.

Более формально, вам дана последовательность из S различных чисел. Переупорядочьте её любым способом. Единственное ограничение — полученная последовательность не должна быть отсортирована ни по возрастанию, ни по убыванию.

Формат входного файла

В первой строке вам дано число T , задающее количество тестов в файле. Перед каждым тестом есть пустая строка. Каждый тест состоит из двух строк. Первая строка содержит число N ($3 \leq N \leq 1000$), задающее длину последовательности. Вторая строка содержит N 32-битных знаковых чисел, разделённых пробелами. Эти числа попарно различны.

Формат выходного файла

Выведите переупорядоченные последовательности в таком же порядке и в таком же формате, как во вводе.

Пример

stdin	stdout
2	2
5	5
1 2 3 4 5	5 1 4 3 2
8	8
3 1 4 47 5 9 2 6	3 1 4 47 5 9 2 6

Разбор задачи В. Антисортировка

В этой задаче достаточно отсортировать массив и поменять первые 2 элемента местами. Так как $n \geq 3$, то после этого мы получим неотсортированную последовательность.

Задача С. Астрид и квадраты

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Маленькая девочка Астрид любит вырезать из бумаги различные фигуры. Особенно ей нравятся квадраты.

На столе перед Астрид лежит бумажный прямоугольник размера $t \times n$ сантиметров. Девочка хочет, чтобы на столе остались одни лишь квадраты. Пока это не так, она берёт со стола прямоугольник и одним прямолинейным разрезом отрезает от него квадрат. После этого квадрат остаётся на столе, а с

остатком происходит то же самое: если он не квадратный, от него одним прямолинейным разрезом отрезается квадрат, и так далее. Наконец, после того, как очередное разрезание привело к появлению двух квадратов, они оба кладутся на стол, и разрезания заканчиваются.

Сколько квадратов окажется у Астрид на столе, когда она закончит разрезания?

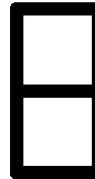
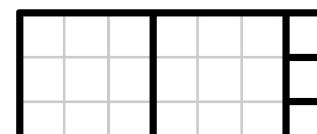
Формат входного файла

В единственной строке ввода заданы целые числа m и n — стороны исходного прямоугольника в сантиметрах ($1 \leq m, n \leq 1\,000\,000\,000$).

Формат выходного файла

Выведите одно число — количество квадратов, которые окажутся на столе после всех разрезаний.

Примеры

stdin	stdout	explanation
1 2	2	
7 3	5	

Пояснение

В пояснениях справа от примеров показан исходный прямоугольник. Он разделён на квадраты, которые окажутся на столе после всех разрезаний.

В первом примере от прямоугольника 1×2 отрезается квадрат 1×1 . Оставшийся прямоугольник 1×1 также является квадратом, поэтому разрезания заканчиваются.

Во втором примере от прямоугольника 7×3 отрезается квадрат 3×3 , и остаётся прямоугольник 4×3 . От него отрезается ещё один квадрат 3×3 , остаётся прямоугольник 1×3 . От этого прямоугольника отрезается квадрат 1×1 , остаётся прямоугольник 1×2 . Наконец, при разрезании этого прямоугольника получается ещё два квадрата 1×1 .

Разбор задачи С. Астрид и квадраты

Будем честно эмулировать разрезания. Получим следующую формулу:

$$f(n, m) = \left\lfloor \frac{n}{m} \right\rfloor + f(n \bmod m, m)$$

(эта формула работает если $n \geq m$). Если честно ее вычислять, то время работы будет логарифмическое(по аналогии с алгоритмом Евклида).

Задача D. Берт и землеройки

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Маленький мальчик Берт любит изучать поведение животных. В этот раз он наблюдает за поведением стайки землероек.

Берт поймал n землероек и выпустил их стайкой на лужайку перед своим домом. Как известно, землеройки предпочитают индивидуальный образ жизни, поэтому они стремятся скрыться от Берта и друг от друга как можно скорее. Каждую секунду последовательно происходят два события:

1. В начале секунды каждая стайка, в которой больше одной землеройки, разделяется ровно на две стайки. В каждой получившейся стайке должна быть хотя бы одна землеройка.
2. В конце секунды одна землеройка из каждой стайки прячется, зарывшись в траву.

Изначально все землеройки находятся в одной стайке. От того, как землеройки делятся на стайки в начале каждой секунды, зависит, сколько секунд пройдёт, прежде чем все они спрячутся. Какое минимальное и максимальное количество секунд может пройти от начала эксперимента, прежде чем все землеройки спрячутся?

Формат входного файла

В единственной строке ввода задано целое число n — количество землероек в начале эксперимента ($1 \leq n \leq 1\,000\,000\,000$).

Формат выходного файла

Выведите два числа, разделив их пробелом — минимальное и максимальное количество секунд, которое может пройти, прежде чем все землеройки спрячутся.

Примеры

stdin	stdout	explanation
2	1 1	$2 = 1 + 1 \xrightarrow{t=1}$ $2 = 1 + 1 \xrightarrow{t=1}$
5	2 3	$5 = 3 + 2 \xrightarrow{t=1} 2 + 1 = 1 + 1 + 1 \xrightarrow{t=2}$ $5 = 4 + 1 \xrightarrow{t=1} 3 = 2 + 1 \xrightarrow{t=2} 1 = 1 \xrightarrow{t=3}$

Пояснение

В пояснениях справа от примеров показаны варианты разделения землероек на стайки. В первой строке показан один из возможных вариантов, позволяющих землеройкам спрятаться за минимальное количество секунд, а во второй — за максимальное. Выражения вида $a_1 + a_2 + \dots = b_1 + b_2 + \dots$ означают, что в результате разделения стаек из a_1, a_2, \dots землероек образовались стайки из b_1, b_2, \dots землероек. Стрелка $\xrightarrow{t=x}$ означает конец x -й секунды. В этот момент одна землеройка из каждой стайки прячется, зарывшись в траву.

В первом примере стайка из двух землероек в начале первой секунды разделится на две стайки по одной землеройке, а в конце первой секунды обе землеройки спрячутся.

Во втором примере стайка из пяти землероек может в начале первой секунды разделиться на 3 и 2 землеройки, а может на 4 и 1 землеройку.

В первом случае в конце первой секунды на лужайке останется две стайки: из 2 землероек и из 1 землеройки. Первая из них в начале второй секунды разделится, и в конце второй секунды все три оставшиеся землеройки спрячутся.

Во втором случае в конце первой секунды осталась одна стайка из трёх землероек. В начале второй секунды она разделится на 2 и 1 землеройку. В конце второй секунды спрячутся все землеройки, кроме одной. Эта последняя землеройка спрячется в конце третьей секунды.

Разбор задачи Д. Берт и землеройки

Максимального времени можно добиться, если группа будет преобразовываться так: $n \rightarrow (n - 1, 1) \rightarrow n - 2$. Тогда ответом будет $\lfloor \frac{n+1}{2} \rfloor$.

Минимального времени можно добиться, если группа будет преобразовываться так: $n \rightarrow (\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil) \rightarrow (\lfloor \frac{n}{2} \rfloor - 1, \lceil \frac{n}{2} \rceil - 1)$. Тогда ответом будет $T(n) = 1 + T(\lceil \frac{n}{2} \rceil - 1) = \lfloor \log_2(n + 1) \rfloor$.

Задача E. Марсианская архитектура

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Кролик Крис нашел следы древней цивилизации Марса. В небольшой телескоп отважному астроному удалось увидеть архитектурный шедевр — “Дорогу к Солнцу”. Это сооружение состоит из одинаковых по размеру кубических камней. Фундамент разбивает всю “дорогу” на ячейки, в которые плотно ложатся кубокамни. Таким образом, каждой ячейке фундамента можно присвоить координату. Для того, чтобы стать вождём, марсианин должен проложить путь к Солнцу, то есть построить из этих кубокамней на заданном фундаменте лестницу. Лестницу можно описать количеством камней в начальной координате и координатами начала и конца лестницы. Каждая следующая ячейка в направлении возрастания координаты должна содержать на один кубокамень больше, чем предыдущая. Причём, если в ячейке уже были до этого камни — в текущем строительстве они не считаются, лестницу просто строили поверх них. Другими словами, пусть строится лестница с координатой начала l , координатой конца r и количеством камней в начальной координате x . Это означает, что в ячейке l добавится x камней, в $l + 1$ добавится $x + 1$ камней, ..., в ячейке r добавится $x + r - l$ камней.

Крису удалось отыскать древний манускрипт, содержащий описания всех лестниц. Теперь он хочет сравнить эти данные, чтобы быть уверенным в том, что нашёл именно “Дорогу к Солнцу”. Для этого он выбрал некоторые ячейки дороги и посчитал суммарное количество кубокамней, которые накопились за всю марсианскую историю, а потом попросил вас с помощью манускрипта вычислить, чему должна быть равна эта сумма, для проверки.

Формат входного файла

Первая строка содержит три целых числа, разделённых пробелами: n , m и k ($1 \leq n, m \leq 10^5$, $1 \leq k \leq \min(100, n)$) — количество ячеек, количество “Дорог к Солнцу” и количество запросов соответственно. Каждая из последующих m строк содержит по три целых числа, разделённых пробелами: a_i , b_i и c_i ($1 \leq a_i \leq b_i \leq n$, $1 \leq c_i \leq 1000$) — описание лестницы, содержащее координаты её начала и конца, а также высоту начальной ячейки. Далее следует строка, содержащая k различных целых чисел b_i , разделённых пробелами. Все эти числа в пределах от 1 до n — ячейки, количество камней в которых интересует Криса.

Формат выходного файла

Требуется вывести в единственной строке одно число — суммарное количество камней во всех интересующих Криса ячейках.

Примеры

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

Разбор задачи Е. Марсианская архитектура

Для того, чтобы добавить к подотрезку $[l, r]$ арифметическую прогрессию можно посмотреть на разности между соседними элементами и посмотреть, в каких точках эти разности меняются. В точке l делается $+1$ к разности, в точке $r + 1$ делается -1 к разности. Тогда мы просто идем слева направо по массиву и накапливаем текущую разность(это на самом деле аналог производной) и вычисляем значение в точке через прошлое значение и текущую разность.

Задача F. Камилла и язык программирования WR

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 256 Мб

Маленькая девочка Камилла любит программировать. Она придумала и реализовала свой собственный язык программирования.

Этот язык называется “WR”, и он работает с *wr-строками* — строками, состоящими только из букв “w” и “r”. Программа на этом языке — это wr-строка, результат работы программы также является wr-строкой.

В языке WR всего две команды. Команда записи (`write`) начинается с маленькой буквы “`w`” и состоит из двух символов: первый из них — сама буква “`w`”, а второй — буква, которую нужно вывести. Команда повтора (`repeat`) состоит из одного символа — маленькой буквы “`r`” — и означает, что нужно один раз повторить выполнение предыдущей команды.

При выполнении programma разбивается на команды слева направо, а затем команды выполняются последовательно. Если что-то из этого невозможно, вся programma считается некорректной. Это происходит в двух случаях:

1. Очередная команда начинается с буквы “`w`”, но второго символа в этой команде нет, так как эта буква “`w`” — последняя буква программы.
2. Команда повтора — первая в программе, поэтому она не может повторить предыдущую команду.

Результат работы некорректной programma — пустая строка.

Камилле нравится, что на её языке можно написать программу, которая выведет другую программу, которую можно тоже выполнить и получить третью программу, и так далее. Девочка хочет последовательно выписать все программы, которые получились при таком процессе. Сначала она выбрала некоторую исходную `wr`-строку и объявила её *исполняемой*. После этого Камилла выполняет следующий алгоритм:

1. Выписывает исполняемую строку.
2. В результате работы исполняемой строки как программы на языке WR получает новую строку.
3. Если полученная новая строка пуста, заканчивает работу, а иначе объявляет эту новую строку исполняемой и вновь переходит к шагу 1.

По заданной исходной `wr`-строке выясните, какие строки и в каком порядке выписала Камилла.

Формат входного файла

В единственной строке ввода задана `wr`-строка — последовательность букв “`w`” и “`r`”. Она содержит от 1 до 100 букв и не содержит пробелов.

Формат выходного файла

Выведите все строки, которые выписала Камилла, в порядке их выписывания.

Примеры

stdin	stdout	explanation
wwrrwrr	wwrrwrr wwwrr wrr rr	write w , repeat, repeat, write r , repeat write w , write r , repeat write r , repeat repeat <ERROR>
wwrwr	wwrwr wwr ww w	write w , repeat, write r write w , repeat write w write <ERROR>

Разбор задачи F. Камилла и язык WR

В данной задаче нужно честно проэмулировать выполнения программы на языке WR.

Задача G. Ворчливые коровы

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

У фермера Джона N коров. Каждый вечер они выстраиваются в очередь, чтобы их подоили. У каждой коровы есть показатель ворчливости — целое число. Показатели ворчливости всех коров попарно различны.

Ворчливые коровы могут повредить оборудование фермы, поэтому фермер Джон хочет переупорядочить их так, чтобы доить коров в порядке возрастания их ворчливости. Фермер может выбрать двух любых коров (не обязательно стоящих рядом) с ворчливостями X и Y и поменять их местами за время $X + Y$. Эту операцию можно повторять последовательно сколько угодно раз.

Помогите фермеру Джону вычислить минимальное время, которое потребуется для того, чтобы переставить коров в очереди таким образом, чтобы доить их в порядке возрастания ворчливости.

Формат входного файла

В первой строке ввода задано целое число N — количество коров ($1 \leq N \leq 10\,000$). Следующие N строк содержат по одному числу каждая. Это ворчливости коров в порядке очереди на дойку. Гарантируется, что ворчливости — попарно различные целые числа от 1 до 100 000.

Формат выходного файла

Выведите одно число — минимальное время, за которое можно переупорядочить коров в порядке возрастания ворчливости.

Пример

stdin	stdout
3	7
2	
3	
1	

Пояснение

В примере три коровы. Изначально первой стоит корова с ворчливостью 2, второй — с ворчливостью 3, а третьей — с ворчливостью 1.

Фермер Джон может сначала поменять коров с ворчливостями 1 и 3 за время $1 + 3 = 4$ и из порядка 2 3 1 получить порядок 2 1 3. Затем фермер может поменять коров с ворчливостями 1 и 2 за время $1+2 = 3$ и получить требуемый порядок 1 2 3.

Разбор задачи G. Ворчливые коровы

Рассмотрим стартовую перестановку коров. Выделим в ней циклы. Рассмотрим один цикл длины L и попытаемся его отсортировать за минимальную стоимость. Пусть стоимость i -й коровы равна C_i . Понятно, что мы должны сделать хотя бы $L - 1$ обмен. Рассмотрим два элемента в цикле. Тогда если мы их поменяем местами, то цикл распадется на два цикла длинами L_1 и L_2 , такими что $L_1 + L_2 = L$. Заметим, что таким образом каждый элемент нам придется хотя бы раз с чем-то поменять. Тогда давайте делать так: меняем местами минимальный элемент на цикле с его соседом так, чтобы остался цикл длины $L - 1$, и платим мы $C_{min} + C_i$. Таким образом мы заплатим минимальную стоимость $(\sum_{i=1}^L (C_i + C_{min})) - 2C_{min}$, чтобы отсортировать перестановку, состоящую из одного цикла. Но теперь вспомним, что у нас есть еще и другие циклы. Заметим, что если у нас есть два цикла, то при перемене элементов из этих циклов мы получим один большой цикл, причем мы это можем сделать за сумму значений минимальных элементов из циклов. Тогда получаем жадное решение общей задачи — найти все циклы, затем выделить цикл с самым минимальным элементом. Теперь для каждого цикла мы смотрим что выгоднее — развернуть его описанным выше способом или приклеить его к циклу с минимальным элементом, а потом развернуть в конце весь минимальный цикл.

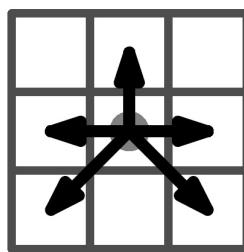
Задача Н. Давид и осёл

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Маленький мальчик Давид любит играть в настольные игры. Научившись играть в шахматы и шашки, он решил придумать свою фигуру, которая ходит по доске по определённым правилам.

Давиду очень нравится, как ходит шахматный конь, поэтому он захотел придумать что-нибудь похожее. Перебрав в уме известных ему животных, мальчик решил посвятить свою фигуру ослу.

Осёл, придуманный Давидом — это фигура, которая умеет перемещаться по прямоугольной клетчатой доске. Он ходит не так быстро, как шахматный конь. Кроме того, осёл упрям и не ходит в некоторые стороны. Следующая схема показывает все возможные ходы осла.



Центральная клетка схемы — начальное положение, а стрелки показывают, куда осёл может переместиться за один ход: на одну клетку вверх, влево, по диагонали влево-вниз, вправо и по диагонали вправо-вниз. Если осёл находится у края доски, можно делать только те из перечисленных ходов, которые не приводят к выходу за границы доски.

Давид гордится выносливостью своего осла: он может обойти любую прямоугольную доску размера $w \times h$ клеток, побывав в каждой её клетке ровно один раз.

Для заданных размеров доски — ширины w и высоты h — выясните, как это сделать. Разрешается начать маршрут из любой клетки доски. Возвращаться в исходную клетку не нужно.

Формат входного файла

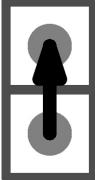
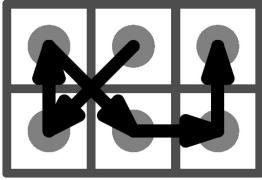
В единственной строке ввода заданы целые числа w и h — ширина и высота доски ($1 \leq w, h \leq 30$).

Формат выходного файла

Выведите $w \cdot h$ строк. В каждой из них выведите два числа, разделив их пробелом — номер столбца x и номер строки y посещённой ослом клетки. Столбцы доски нумеруются слева направо от 1 до w , а строки — сверху вниз от 1 до h . Клетки должны быть перечислены в порядке их посещения. Из каждой выведенной клетки, кроме последней, должно быть возможно за один ход переместить осла в следующую выведенную клетку согласно правилам. Каждая клетка доски должна встречаться в выводе ровно один раз.

Если возможных обходов доски несколько, выведите любой из них.

Примеры

stdin	stdout	explanation
1 2	1 2 1 1	
3 2	2 1 1 2 1 1 2 2 3 2 3 1	

Разбор задачи Н. Давид и осёл

Будем идти снизу вверх — в последнем ряду слева-направо, в предпоследнем справа-налево, в предпредпоследнем опять слева направо, и т.д.

11	12	13	14	15
10	9	8	7	6
1	2	3	4	5

Задача I. Эмма и соты

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 256 Мб

Маленькая девочка Эмма любит наблюдать за пчёлами. Недавно она нашла заброшенный улей, в котором сохранился кусочек рамки с пчелиными сотами.

Пчелиные соты, найденные Эммой, при взгляде сверху выглядят как одинаковые правильные шестиугольники со стороной 3 миллиметра, соединённые по сторонам. Пример того, как могут выглядеть соты, показан на рисунке.

Эмма решила записать, как выглядят найденные соты, чтобы потом нарисовать их. Она заметила, что граница сот — одна непрерывная ломаная линия, не пересекающая и не касающаяся сама себя, и решила, что придумать описание границы будет проще всего. Подумав, девочка решила записывать границу так: мысленно встать на неё, выбрать направление обхода и обойти всю границу, побывав в каждой её точке ровно один раз.

Рассмотрим подробнее этот мысленный эксперимент. Сначала Эмма становится в какой-то угол шестиугольника на границе сот. В этом углу встречаются два отрезка границы. Эмма поворачивается в сторону того из отрезков, для которого получается, что справа от неё соты, а слева — пустота. Затем девочка начинает двигаться по границе. Как можно увидеть на рисунке выше, граница состоит из прямых участков длиной 3 миллиметра, после каждого из которых линия границы поворачивает на 60 градусов либо направо, либо налево. Эмма выписывает последовательность поворотов, обозначая поворот направо заглавной буквой “R”, а поворот налево — заглавной буквой “L”, пока не окажется в исходной точке границы смотрящей в исходном направлении.

По полученной Эммой записи границы выясните, из скольких правильных шестиугольников со стороной 3 миллиметра состоят найденные соты.

Формат входного файла

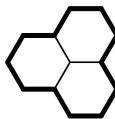
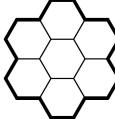
В единственной строке ввода записана последовательность поворотов при обходе границы. Эта строка состоит из букв “L” и “R”, имеет длину от 6 до 100 букв и не содержит пробелов. Гарантируется, что данная последовательность

поворотов корректно задаёт границу некоторых сот в соответствии с условием, и эта граница не пересекает и не касается сама себя.

Формат выходного файла

Выведите одно число — количество правильных шестиугольников со стороной 3 миллиметра, из которых состоят соты, имеющие заданную границу.

Примеры

stdin	stdout	explanation
LRRRLRRRLRRR	3	
RLRRLRRLRRLRRLRRLR	7	

Пояснение

В пояснениях справа от примеров показаны соты, имеющие заданную границу.

В первом примере на границе 12 поворотов, и эта граница ограничивает соты, состоящие из трёх правильных шестиугольников со стороной 3 миллиметра.

Во втором примере на границе 18 поворотов, и эта граница ограничивает соты, состоящие из семи правильных шестиугольников со стороной 3 миллиметра.

Разбор задачи I. Эмма и соты

Можно просто построить многоугольник по обходу, вычислить его площадь и поделить на площадь одного шестиугольника.

Задача J. Побег

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Принцесса собирается сбежать из пещеры дракона. Это нужно как следует

спланировать.

Принцесса бежит со скоростью v_p миль в час, дракон летает со скоростью v_d миль в час. Дракон обнаружит побег через t часов и немедленно бросится в погоню. Затем кажется безнадёжной, но принцесса заметила, что дракон очень жаден и не очень умён. Чтобы задержать его, принцесса решает прихватить с собой несколько драгоценностей из его сокровищницы. Когда дракон догоняет принцессу, она может бросить на землю драгоценность: тогда дракон остановится, подберёт упавшее, вернётся в пещеру и потратит f часов в самой пещере на возвращение вещицы на место и наведение порядка в сокровищнице, после чего вновь отправится в погоню, начиная всё с самого начала.

Предполагая, что принцесса будет бежать по прямой без остановок, выясните, сколько драгоценностей ей нужно взять с собой, чтобы успеть добраться до королевского замка, расположенного на расстоянии c миль от пещеры дракона? Если дракон догоняет принцессу в тот момент, когда она добегает до замка, считается, что она успела скрыться в замке раньше (дополнительная драгоценность не нужна).

Формат входного файла

Входные данные содержат целые числа v_p , v_d , t , f и c ($1 \leq v_p, v_d \leq 100$, $1 \leq t, f \leq 10$, $1 \leq c \leq 1000$), каждое число записано в отдельной строке.

Формат выходного файла

Выведите минимальное количество драгоценностей, необходимое для того, чтобы побег удался.

Примеры

stdin	stdout
1 2 1 1 10	2
1 2 1 1 8	1

Пояснение

В первом примере через час после побега принцесса будет находиться на расстоянии 1 от пещеры, а дракон обнаружит побег. Через 2 часа после по-

бега дракон догонит принцессу на расстоянии 2 от пещеры, и ей нужно будет расстаться с первой драгоценностью. Возвращение в пещеру и возня в со-кровищнице займёт у дракона ещё два часа; за это время принцесса убежит на расстояние 4 от пещеры. Во второй раз дракон догонит принцессу на рас-стоянии 8 от пещеры, и ей понадобится вторая драгоценность, после чего она спокойно добежит до замка.

Второй пример аналогичен первому, но второй раз дракон догонит прин-цессу ровно в тот момент, когда она добежит до замка, и вторая драгоценность ей не понадобится.

Разбор задачи J. Побег

Будем эмулировать действия принцессы и дракона. Считаем время, нуж-ное для того, чтобы дракон долетел до принцессы. В этот момент принцес-са сбрасывает драгоценность, дракон долетает до своей пещеры. Повторяем процесс пока принцесса не добежит до замка.

Задача K. Петя и Java

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Маленький Петя недавно начал посещать кружок по программированию. Естественно, перед ним появилась задача выбрать язык, на котором он будет программировать. После долгих размышлений он понял, что Java — лучший выбор. Главным аргументом в пользу выбора Java было то, что в ней есть це-лочисленный тип данных, позволяющий работать с очень большими числами: `BigInteger`.

Однако, после посещения занятий кружка Петя понял, что не все задачи требуют использования типа `BigInteger`. Как оказалось, в некоторых задачах намного удобнее использовать маленькие типы данных. Поэтому возникает вопрос: “Какой целочисленный тип использовать, если нужно хранить natу-ральное число n ?”

Петя знает лишь пять целочисленных типов:

1. Тип `byte` занимает 1 байт и позволяет хранить числа от -128 до 127 .
2. Тип `short` занимает 2 байта и позволяет хранить числа от $-32\,768$ до $32\,767$.

3. Тип **int** занимает 4 байта и позволяет хранить числа от $-2\ 147\ 483\ 648$ до $2\ 147\ 483\ 647$.
4. Тип **long** занимает 8 байт и позволяет хранить числа от $-9\ 223\ 372\ 036\ 854\ 775\ 808$ до $9\ 223\ 372\ 036\ 854\ 775\ 807$.
5. Тип **BigInteger** позволяет хранить любое целое число, но при этом не является примитивным типом, и операции с ним выполняются гораздо медленнее, чем с перечисленными выше типами.

Для всех указанных выше типов значения границ включаются в диапазон значений.

Из этого списка Петя хочет выбрать самый маленький тип, в котором можно хранить натуральное число n . Так как BigInteger работает гораздо медленнее остальных типов, Петя рассматривает его в последнюю очередь. Помогите ему.

Формат входного файла

В первой строке записано целое положительное число n . Оно состоит не более чем из 100 цифр и не содержит ведущих нулей. Число n не может являться пустой строкой.

Формат выходного файла

Выведите первый тип из списка “**byte**, **short**, **int**, **long**, **BigInteger**”, в котором можно хранить натуральное число n в соответствии с данными, приведёнными выше.

Примеры

stdin	stdout
127	byte
130	short
123456789101112131415161718192021222324	BigInteger

Разбор задачи К. Петя и Java

Проще всего было считать число из input на Java в типе BigInteger и честно сравнить с заданными из условия границами. Если писать решение на C++, то можно было считать числа как строки и сравнивать их сначала по длине, а при равенстве длин как строки.

Задача L. Цепная дробь

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Цепной дробью называется выражение вида

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_m}}}},$$

где все a_k целые, $a_k > 0$ при $k > 0$ и $a_m > 1$, если $m > 0$. Эти, на первый взгляд странные, правила позволяют каждому вещественному числу x единственным образом сопоставить цепную дробь так, чтобы выполнялось равенство

$$x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_m}}}}.$$

Например, цепная дробь для числа $7/18$ выглядит как

$$0 + \frac{1}{2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{3}}}};$$

$$\begin{aligned} 7/18 &= 0 + 7/18 = \\ &= 0 + 1/(18/7) = \\ &= 0 + 1/(2 + 4/7) = \\ &= 0 + 1/(2 + 1/(7/4)) = \\ &= 0 + 1/(2 + 1/(1 + 3/4)) = \\ &= 0 + 1/(2 + 1/(1 + 1/(4/3))) = \\ &= 0 + 1/(2 + 1/(1 + 1/(1 + 1/3))). \end{aligned}$$

Цепную дробь можно записывать как $[a_0; a_1, a_2, \dots, a_m]$; в этой записи $7/18 = [0; 2, 1, 1, 3]$.

Не все числа имеют конечную цепную дробь; так, для числа $\sqrt{2} \approx 1,41421356\dots$ цепная дробь имеет вид $[1; 2, 2, 2, 2, 2, 2, \dots]$ — воспользовавшись равенством $\sqrt{2} - 1 = 1/(\sqrt{2} + 1)$, получаем:

$$\begin{aligned}\sqrt{2} &= 1 + (\sqrt{2} - 1) = \\ &= 1 + 1/(\sqrt{2} + 1) = \\ &= 1 + 1/(2 + (\sqrt{2} - 1)) = \\ &= 1 + 1/(2 + 1/(\sqrt{2} + 1)) = \\ &= \dots\end{aligned}$$

Цепная дробь такого вида называется *периодической*; в данном случае предпериод — это число $a_0 = 1$, а период — повторяющееся бесконечное количество раз число 2. Для удобства записи период цепной дроби можно записывать в круглых скобках; для $\sqrt{2}$ сокращённая запись будет выглядеть как $[1; (2)]$. Можно доказать, что для всякого целого $N \geq 0$ цепная дробь числа \sqrt{N} является конечной, если \sqrt{N} — целое число, и периодической в противном случае.

По данному числу N выведите цепную дробь числа \sqrt{N} в сокращённой записи.

Формат входного файла

В первой строке входного файла записано целое число N ($1 \leq N \leq 1000$).

Формат выходного файла

В первую и единственную строку выходного файла выведите сокращённую запись цепной дроби числа \sqrt{N} . Необходимо как можно более точно соблюдать такой же формат вывода, как в примерах. Гарантируется, что правильный ответ на каждый тест содержит не более 10 000 символов.

Примеры

stdin	stdout
1	[1]
2	[1; (2)]
3	[1; (1, 2)]
4	[2]
5	[2; (4)]
6	[2; (2, 4)]
7	[2; (1, 1, 1, 4)]
76	[8; (1, 2, 1, 1, 5, 4, 5, 1, 1, 2, 1, 16)]

Разбор задачи L. Цепная дробь

Пусть мы раскладываем в цепную дробь число вида $\frac{a\sqrt{n}+b}{c}$, где a, b, c целые числа, $c > 0$. Изначально $a = 1, b = 0, c = 1$. Честно вычислим $a_0 = \left\lfloor \frac{a\sqrt{n}+b}{c} \right\rfloor$. Теперь нам нужно разложить в цепную дробь число $\frac{1}{\frac{a\sqrt{n}+b}{c} - a_0} = \frac{c}{a\sqrt{n}+b-a_0c} = \frac{c(-a\sqrt{n}+b-a_0c)}{(a\sqrt{n}+b-a_0c)(-a\sqrt{n}+b-a_0c)} = \frac{-ac\sqrt{n}+(bc-a_0c^2)}{(b-a_0c)^2-n}$

Мы можем сократить числитель и знаменатель на $\gcd(a_{new}, b_{new}, c_{new})$ и рекурсивно запуститься. Заметим, что если начинать с $(1, 0, 1)$, то все a будут равны 1, а $|b|, |c| \leq 2\sqrt{n}$. А длина цикла ведет себя как $O(\sqrt{n} \cdot \ln n)$. Можно попробовать доказать эти факты самим или найти их в литературе по цепным дробям.

Задача M. Шестерёнки

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Несколько шестерёнок соприкасается с зубчатой осью, одним зубцом каждого. Когда ось поворачивается на один зубец по часовой стрелке, все шестерёнки поворачиваются на один зубец против часовой стрелки. На шестерёнках может быть разное количество зубцов, но на каждой есть один, покрашенный зелёной краской.

Ось поворачивается с постоянной скоростью — на один зубец в секунду по часовой стрелке. Известно, что, если продолжать крутить ось бесконечно долго, в какой-то момент все зелёные зубцы одновременно соприкоснутся с осью. Также для каждой шестерёнки известно, сколько секунд требуется, чтобы из начального положения получить такое, в котором зелёный зубец этой шестерёнки впервые соприкоснётся с осью.

Выясните, в какой момент времени все зелёные зубцы впервые соприкоснутся с осью одновременно.

Формат входного файла

В первой строке входного файла записано число N ($2 \leq N \leq 4$) — количество шестерёнок. Вторая строка содержит N чисел A_1, A_2, \dots, A_N ($5 \leq A_k \leq 50$), записанных через пробел — количество зубцов на шестерёнках. В третьей же строке записано N чисел B_1, B_2, \dots, B_N ($0 \leq B_k < A_k$), также через пробел — это моменты первого соприкосновения зелёного зубца

каждой шестерёнки с осью. Другими словами, зелёный зубец k -й шестерёнки соприкасается с осью в моменты $B_k, B_k + A_k, B_k + 2A_k, B_k + 3A_k, \dots$

Формат выходного файла

В выходной файл выведите одно число — номер секунды, в которую все зелёные зубцы впервые соприкоснутся с осью одновременно.

Примеры

stdin	stdout
2 5 6 0 0	0
3 5 6 7 1 2 3	206
4 50 50 50 40 25 25 25 25	25

Разбор задачи М. Шестеренки

Эта задача является простым вариантом китайской теоремы об остатках. Из нее следует, что если существует ответ, то он не превосходит наименьшего общего кратного модулей. Это означает, что можно просто перебрать все варианты и проверить.

Задача N. Задача на НОК

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 256 Мб

Несколько дней назад я узнал, что существует такая штука, как наименьшее общее кратное (НОК). Теперь я часто играю с этим понятием — хочу сделать большое число с помощью НОК.

Но я не хочу использовать слишком много чисел, поэтому я выберу три целых положительных числа (не обязательно различных), каждое из которых не превышает n . Помогите мне найти максимально возможное наименьшее общее кратное таких трёх целых чисел.

Формат входного файла

В первой строке записано целое число n ($1 \leq n \leq 10^6$) — максимальное число, которое можно выбрать.

Формат выходного файла

Выведите единственное целое число — максимально возможное наименьшее общее кратное трёх не обязательно различных целых чисел, каждое из которых не превосходит n .

Примеры

stdin	stdout
9	504
7	210

Пояснение

Наименьшее общее кратное нескольких положительных целых чисел — это наименьшее положительное целое число, кратное им всем.

Результат может получиться достаточно большим. Возможно, 32-битного целого числа не будет достаточно для его хранения. Поэтому рекомендуется использовать 64-битные целые числа.

В последнем примере мы можем выбрать числа 7, 6, 5, их НОК равен $7 \cdot 6 \cdot 5 = 210$. Это — максимальный НОК, который мы можем получить.

Разбор задачи N. НОК

Понятно, что ответ равен $n(n - 1)(n - 2)$, если $\gcd(n, n - 1) = 1$. Из этого соображения следует, что в ответе $a \approx b \approx c \approx n$. Можно перебрать все тройки чисел в интервале $[n - 10, n]$ и получить ОК, но можно написать полный перебор всех вариантов в порядке убывания с отсечениями по ответу, который будет работать примерно за $O(n)$ из-за того, что мы уже имеем отличное стартовое приближение к ответу.

Задача O. Сон Студента

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Как утверждает статистика, студенты спят не более трёх часов в сутки. Но

даже в мире грёз, мерно посыпывая, они не всегда могут избавиться от чувства нависшей угрозы.

Спит Бедный Студент, и снится ему, как сидит он на экзамене по математическому анализу у самого страшного преподавателя современности, трёхкратного героя Советского Союза, лауреата нобелевской премии по отчислению студентов, многоуважаемого профессора Петра Палыча.

Ни на один вопрос не смог ответить Бедный Студент. Значит, вместо большого просторного офиса придётся ему идти работать на ториевые шахты. Но подождите! Пётр Палыч решил дать Студенту последний шанс! Да, такое может быть только во сне.

И профессор начал: “Встретились венерянская девочка и марсианский мальчик на Земле и захотели прогуляться, держась за руки. Но вот беда: у девочки на левой руке a_l пальцев, а на правой — a_r пальцев. У мальчика, соответственно, b_l и b_r . Им будет удобно держаться за руки только тогда, когда никакая пара пальцев девочки не будет соприкасаться между собой, то есть между любыми двумя пальцами девочки есть палец мальчика. И вместе с этим, никакая тройка пальцев мальчика не должна соприкасаться между собой. Определите, смогут ли они взяться за руки так, чтобы обоим было удобно?”

Мальчику и девочке всё-равно, кто пойдёт слева, а кто справа. Разница лишь в том, что, если мальчик пойдёт слева от девочки, то он возьмёт своей правой рукой её левую руку, а если он пойдёт справа — то наоборот.

Формат входного файла

В первой строке даны два целых положительных числа, не превышающие 100: количество пальцев на левой и правой руках венерянской девочки соответственно. Во второй строке даны два натуральных числа, не превышающие 100: количество пальцев на левой и правой руках марсианского мальчика соответственно.

Формат выходного файла

Выведите “YES” или “NO”: ответ на вопрос Петра Палыча.

Примеры

stdin	stdout
5 1 10 5	YES
4 5 3 3	YES
1 2 11 6	NO

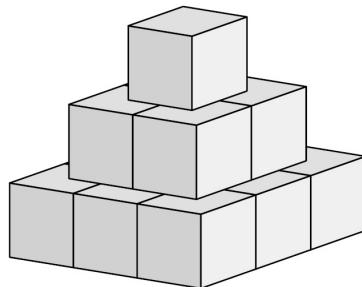
Разбор задачи О. Сон студента

Нужно честно попробовать 2 варианта расположения мальчика и девочки. Они могут ходить вместе, если $d - 1 \leq m$ и $m \leq 2(d + 1)$ (m - число пальцев на руке мальчика, d - число пальцев на руках девочки).

Задача Р. Постройка пирамиды

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Великий Фараон Эхнатон, Сын Солнца, повелел построить новую пирамиду. Для постройки были доставлены n каменных блоков, имеющих форму куба пяти локтей в длину, пяти в ширину и пяти в высоту (локоть — древнеегипетская мера длины, равная примерно 466 миллиметрам). Пирамида строится так: сначала выкладывается основание $k \times k$ блоков, на него кладётся следующий ярус размера $(k - 1) \times (k - 1)$ блоков, и так далее до последнего k -го яруса, состоящего всего лишь из одного каменного блока. Пирамида высоты в три каменных блока, состоящая из $9 + 4 + 1 = 14$ блоков, показана на рисунке.



Главный землемер фараона хочет узнать, какова максимальная высота пирамиды, которую можно построить из n блоков — не зная заранее высоты, он не может начать строительство. К сожалению, эта задача оказалась ему не по силам, и он обратился за помощью к вам — своему помощнику.

Формат входного файла

В первой строке входного файла записано одно целое число n — количество каменных блоков ($1 \leq n \leq 32\,000$).

Формат выходного файла

В первой строке выходного файла выведите одно целое число — максимальную возможную высоту пирамиды в блоках.

Примеры

stdin	stdout
9	2
14	3

Разбор задачи P. Постройка пирамиды

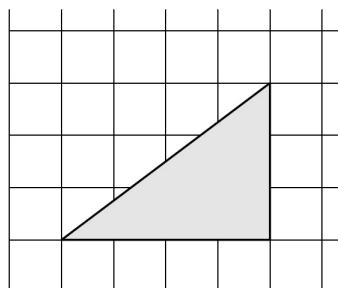
Можно просто по очереди снимать слои.

Задача Q. Треугольная комната

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

После укладки основания пирамиды (это оказался квадрат из 45×45 блоков) пришла пора вырубить в ней комнату — будущую усыпальницу для одного из родственников фараона. Эхнатон решил, что усыпальница в этот раз, вопреки всем канонам, будет треугольной. Стороны её должны иметь длины, кратные размерам блоков, из которых построена пирамида. Одна сторона должна быть в длину как a блоков, стоящих в ряд друг за другом, другая — как b блоков, третья — как c блоков. Числа a , b и c — это три знаменательных даты в жизни родственника фараона.

Главный землемер, руководящий строительством, хочет, кроме того, чтобы каждый из трёх углов комнаты находился на стыке четырёх блоков, из которых составлена пирамида. Иными словами, если нарисовать основание пирамиды на плоскости и ввести координаты так, чтобы один угол пирамиды оказался в начале координат, а противоположный — в точке $(45, 45)$, то вершины треугольника, соответствующего комнате, должны оказаться в точках с целыми координатами. Кроме того, углы комнаты должны быть строго внутри основания пирамиды. Пример такой комнаты для $a = 3$, $b = 5$ и $c = 4$ показан на рисунке.



Помогите главному землемеру и в этот раз — выясните, каким образом вырубить комнату с данными длинами сторон так, чтобы выполнить его требования, или укажите, что это невозможно.

Формат входного файла

В первой строке входного файла записаны через пробел три целых числа a , b и c — длины стен треугольной комнаты ($1 \leq a, b, c \leq 43$). Числа a , b и c таковы, что из трёх отрезков с такими длинами можно составить треугольник, площадь которого строго положительна.

Формат выходного файла

Если комнату с данными длинами стен указанным образом поместить в пирамиду невозможно, в первой строке выходного файла выведите “**NO**”. В противном случае в первой строке выведите “**YES**”, а в следующих трёх строках — координаты углов комнаты, по одному углу на строке. Порядок углов не имеет значения. Все координаты должны быть целыми числами в промежутке от 1 до 44, включительно.

Примеры

stdin	stdout
3 5 4	YES 1 1 5 1 5 4
2 3 4	NO

Разбор задачи Q. Треугольная комната

Можно просто перебрать три вершины и проверить длины сторон на совпадение с длинами из условия.

Задача R. SMS

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 256 Мб

Маленький морж Клычок, как и все современные моржи, любит общаться с помощью SMS. Однажды он столкнулся с проблемой: при отправке больших текстов они разделяются на части по n символов (размер одного SMS сообщения), и разрываются целые предложения или даже слова!

Клычку это не понравилось, и перед ним возникла задача разбить текст на сообщения самостоятельно таким образом, чтобы ни одно предложение не было разбито на части при отправке, а количество SMS сообщений для отправки было бы минимальным. Если два подряд идущих предложения находятся в разных сообщениях, то пробел между ними можно не учитывать (Клычок этот пробел не набирает).

Текст маленького моржа имеет следующий вид:

```
TEXT ::= SENTENCE | SENTENCE SPACE TEXT
SENTENCE ::= WORD SPACE SENTENCE | WORD END
END ::= {'.', '?', '!' }
WORD ::= LETTER | LETTER WORD
LETTER ::= {'a'...'z', 'A'...'Z'}
SPACE ::= ' '
```

Под **SPACE** следует подразумевать символ пробела.
Сколько же сообщений отправил Клычок?

Формат входного файла

В первой строке задаётся целое число n — размер одного сообщения ($2 \leq n \leq 255$). В следующей строке находится сам текст. Длина текста не превышает 10^4 символов. Гарантируется, что текст удовлетворяет описанному выше формату. В частности, из этого следует, что текст не пуст.

Формат выходного файла

В первой и единственной строке выведите количество SMS сообщений, которое потребуется Клычку. Если разбить текст невозможно, выведите “**Impossible**” без кавычек.

Примеры

stdin	stdout
25 Hello. I am a little walrus.	2
2 How are you?	Impossible
19 Hello! Do you like fish? Why?	3

Пояснение

Рассмотрим третий пример. Текст будет разбит на три сообщения: “Hello!”, “Do you like fish?” и “Why?”.

Разбор задачи R. SMS

Нужно аккуратно разбить текст на предложения и жадно распихать предложения по смскам.

Задача S. Неквадраты

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Задано целое положительное число n . Выясните, может ли оно быть представлено в виде произведения k целых чисел, ни одно из которых *не* является квадратом целого числа.

Формат входного файла

Первая строка ввода содержит целое число t ($1 \leq t \leq 10$) — количество тестовых случаев. Каждая из последующих t строк содержит один тестовый случай, состоящий из двух целых чисел n ($1 \leq n \leq 1\,000\,000\,000$) и k ($2 \leq k \leq 50$).

Формат выходного файла

Для каждого тестового случая выведите в отдельной строке слово “YES” в том случае, если существует такой набор из k целых чисел a_i , что $n = a_1 \cdot a_2 \cdots a_k$ и ни одно из a_i не является квадратом целого числа, и слово “NO” в противном случае.

Пример

stdin	stdout
2	NO
1 3	YES
7 2	

Разбор задачи S. Неквадраты

Самое главное в этой задаче — понять второй семпл:

$$7 = -1 \times -7.$$

А дальше понятно, что если k четное, то можно сделать все множители отрицательными (в качестве множителей, например могут выступать само n и $k - 1$ единица) и все будет хорошо. При нечетном k мы понимаем, что 1 никак нельзя представить требуемым образом. Оставшиеся числа можно представить так — взять любой простой делитель p числа n , $-\frac{n}{p}$ и $(k - 2)$ минус единицы.

Задача T. Перестановка цифр

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Нет ничего прекраснее, чем целое число.

Вам задано целое число n . Запишите его в десятичной записи без ведущих нулей. Пусть M — количество цифр в получившейся записи. Далее следует ровно k раз выполнить один шаг следующего вида:

Выберите i и j — две различные позиции в записи числа так, что $1 \leq i < j \leq M$ (цифры нумеруются с первой). Поменяйте местами цифры на этих двух позициях. Если после шага в числе появляются ведущие нули (то есть цифра на позиции j равна нулю, а $i = 1$), такой шаг делать нельзя.

Выведите наибольшее возможное число, которое может получиться после выполнения описанной процедуры. Если ровно k шагов сделать невозможно, выведите -1 .

Формат входного файла

В первой строке ввода задано два целых числа n и k ($1 \leq n \leq 1\,000\,000$, $1 \leq k \leq 10$).

Формат выходного файла

Выведите одно целое число — ответ на задачу.

Примеры

stdin	stdout
16375 1	76315
432 1	423
90 4	-1
5 2	-1

Пояснение

В первом примере оптимально будет поменять местами цифры 1 и 7.

Во втором примере результат получается даже меньше, чем исходное число.

В третьем примере мы не можем сделать ни одного шага, поскольку в результате любого шага в числе появятся ведущие нули.

В четвёртом примере мы не можем выбрать две различных позиции для шага.

Разбор задачи Т. Перестановка цифр

При прочтении условия сразу хочется написать жадный алгоритм перемены цифр местами, но к сожалению он неверен. Правильным решением является динамика $dp[n][k]$ — можно ли получить число n за k шагов. Переход осуществляется за $\log_{10}^2(n)$ простым перебором того, что нужно менять местами. Так как мы всего лишь меняем цифры местами, то состояний у нас будет не nk , а $(\log_{10}(n))!k$ и можно их хранить не все, а только текущие достижимые состояния при фиксированном k .

Задача U. Пристрастный учитель

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

По итогам учебного года учитель решил поощрить своих учеников — раздать им немного ирисок. Он попросил n учеников встать в ряд. Поскольку учитель очень пристрастен, он руководствуется следующим правилом при раздаче ирисок.

Он смотрит на первых двух учеников и даёт больше ирисок тому из них, у которого выше оценки. Если у этих двух учеников одинаковые оценки, то они

получают одинаковое количество ирисок. Процесс раздачи проходит аналогичным образом для любой пары рядом стоящих учеников, начиная с первого и заканчивая последним.

Известно, что каждый ученик получит как минимум одну ириску. Вам следует определить, сколько ирисок учитель может дать каждому ученику, таким образом, чтобы общее количество разданных ирисок было наименьшим.

Формат входного файла

В первой строке ввода записано количество учеников n ($2 \leq n \leq 1000$). Во второй строке содержится $(n - 1)$ знаков, каждый из которых — “L”, “R” или “=”. Для пары рядом стоящих учеников “L” означает, что у ученика слева оценки выше, “R” означает, что выше оценки у ученика справа, а “=” означает, что их оценки одинаковые. Первый знак соответствует отношению между оценками первого и второго учеников, второй знак — между оценками второго и третьего, и так далее. Первый ученик в ряду считается стоящим слева от всех, последний — справа.

Формат выходного файла

Выведите n целых — сколько ирисок получит каждый ученик в ряду, начиная с первого и заканчивая последним.

Примеры

stdin	stdout
5 LRLR	2 1 2 1 2
5 =RRR	1 1 2 3 4

Разбор задачи U. Пристрастный учитель

Раздадим изначально каждому ученику по одной конфете, а потом будем повторять следующую операцию пока процесс не стабилизируется: находим место, где не выполняются условия и дадим конфеты ученикам так, чтобы неравенства выполнялись. Понятно, что суммарно мы дадим не более, чем n^2 конфет. Получаем очень простое решение за $O(n^3)$.

Альтернативное решение: пусть у первого ученика будет x конфет. Тогда мы можем вычислить число конфет у каждого из учеников. Останется только найти такое минимальное x , что у каждого будет хотя бы одна конфета.

Задача V. Поезд

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 256 Мб

Вася живет посередине Программистской ветки метро. У него есть две девушки: Даша и Маша, живущие на разных концах ветки, каждая не ведающая о существовании другой.

Когда у Васи появляется свободное время, он едет к одной из своих девушек. Он спускается в метро в некоторый момент времени, ждёт первого поезда и едет на нём до конца ветки к соответствующей девушке. Однако поезда ходят с разной частотой: в Дашином направлении станцию проезжает один поезд в a минут, а в Машином — один поезд в b минут. В случае, если поезда подошли одновременно, Вася едет в направлении с меньшей частотой хождения поездов, то есть к той девушке, в чью сторону поезда ходят реже (см. пояснение к третьему примеру).

Известно, что поезда начинают своё хождение одновременно до появления Васи. То есть расписание поездов таково, что существует момент времени, когда два поезда приезжают одновременно.

Помогите Васе посчитать, к какой девушке он будет попадать чаще.

Формат входного файла

В первой строке записано два целых числа a и b ($a \neq b$, $1 \leq a, b \leq 10^6$).

Формат выходного файла

Выведите “Dasha”, если Вася будет чаще попадать к Даше, “Masha”, если к Маше, и “Equal”, если одинаково часто к обеим девушкам.

Примеры

stdin	stdout
3 7	Dasha
5 3	Masha
2 3	Equal

Пояснение

Разберём третий пример.

Пусть поезда начали своё хождение в нулевой момент времени. Понятно, что моменты прибытия поездов будут периодичны с периодом 6. Поэтому достаточно показать, что, спускаясь в метро в момент времени внутри полуинтервала $(0, 6]$, Вася будет попадать к обеим девушкам одинаково часто.

Если Вася спустился в момент от 0 до 2, он уезжает к Даше на поезде, прибывающем ко второй минуте.

Если Вася спустился в момент от 2 до 3, он уезжает к Маше на поезде, прибывающем к третьей минуте.

Если Вася спустился в момент от 3 до 4, он уезжает к Даше на поезде, прибывающем к четвёртой минуте.

Наконец, если Вася спустился в момент от 4 до 6, то он дожидается прибытия обоих поездов к шестой минуте и уезжает к Маше, потому что в её направлении поезда ходят реже.

Суммарно на Дашу и Машу приходится поровну — по три минуты, значит, к обеим девушкам он будет попадать одинаково часто.

Разбор задачи V. Поезда

В принципе понятно, что Вася будет чаще ездить к той девушке, к которой чаще ездят поезда. В нашем случае это почти так — сначала заметим, что если сократить a и b на одно и тоже число, то ответ не поменяется. Несложно доказать, что если a и b взаимно просты и отличаются на 1, то ответ Equal. Иначе действует наше изначальное предположение. Но в задаче можно было честно проэмулировать процесс до НОК(a, b) за $O(a + b)$.

Задача W. Ловушки

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Эхнатон, Сын Солнца, хочет защитить новую пирамиду от воров и мародёров. Для этого в пирамиде, кроме усыпальницы, будет вырублено ещё несколько комнат поменьше. Все комнаты, включая усыпальницу, будут соединены системой коридоров таким образом, чтобы из любой комнаты в любую другую существовал ровно один путь, не проходящий ни по какому коридору дважды. Во многих комнатах будут установлены смертоносные ловушки.

У Эхнатона есть верный слуга — шаман родом из далёких южных стран, владеющий магией вуду. Фараон повелел шаману наложить заклятие на пирамиду: как только непрошеный посетитель оказывается в тупике — комнате, в которую ведёт только один коридор — волны отчаяния и безысходности должны захлестнуть его разум и подтолкнуть несчастного к ближайшей ловушке. Заклятие, однако, не должно действовать в священной комнате — усыпальнице, — даже если из неё ведёт всего один коридор.

Чтобы наложить заклятие, шаману необходимо знать точное количество тупиков в пирамиде. Чтобы узнать это число, он позвал вас — помощника главного землемера. К счастью, у вас сохранился план пирамиды. Ответьте шаману, сколько тупиков насчитывается на этом плане.

Формат входного файла

В первой строке входного файла записаны через пробел два целых числа n и m — количество комнат и количество коридоров, соответственно ($2 \leq n \leq 20, m > 0$). Следующие m строк содержат описания коридоров; в i -й из этих строк записаны два числа u_i и v_i через пробел, означающие, что i -й коридор соединяет комнаты с номерами u_i и v_i ($1 \leq u_i, v_i \leq n, u_i \neq v_i$). Комнаты нумеруются с единицы; комната с номером 1 — это усыпальница. Любые две комнаты соединены не более чем одним коридором.

Формат выходного файла

В первой строке выходного файла выведите одно целое число — количество тупиков. Помните, что усыпальница тупиком не считается.

Примеры

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

Разбор задачи W. Ловушки

Нужно просто посчитать у каждой вершины степень и выдать те вершины у которых степень равна 1, кроме корня.

Задача X. Магия вуду

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 256 Мб

Чтобы наложить на пирамиду заклятие, шаман Эхнатона должен использовать один или несколько магических кристаллов. Каждый кристалл харак-

теризуется целым числом — своей силой. От силы используемых кристаллов зависит мощность заклятия: при использовании k кристаллов, имеющих силы f_1, f_2, \dots, f_k , заклятие будет иметь силу $(f_1 \cdot f_2 \cdot \dots \cdot f_k) \bmod m$, где m — магическое число вуду, равное $2^{30} = 1\,073\,741\,824$. Выражение “ $a \bmod b$ ” означает остаток от деления a на b .

Шаман уже убедился в ваших способностях к вычислениям, и дал новое задание. Он снабдил вас информацией об имеющихся у него магических кристаллах. Теперь шаман просит сказать ему, какой максимальной мощности может достичь заклятие при правильном выборе из имеющихся кристаллов одного или нескольких, которые надо использовать.

Формат входного файла

В первой строке входного файла записано целое число n — количество магических кристаллов, имеющихся в распоряжении шамана ($1 \leq n \leq 12$). Во второй строке записаны n целых чисел p_1, p_2, \dots, p_n через пробел — силы магических кристаллов ($1 \leq p_i \leq 10\,000$).

Формат выходного файла

В первой строке выходного файла выведите одно число — максимальную возможную силу заклятия.

Примеры

stdin	stdout
2	3
1 3	
3	99990000
10000 11 9999	

Разбор задачи X. Магия вуду

Переберем за 2^n какие числа мы возьмем и честно вычислим максимум. Понятно, что нужно умножать и брать по модулю на каждом шаге, а не в конце, чтобы избежать работы с большими числами.

Задача на самое короткое решение. Суперминимум

Вход:

Выход:

Ограничение по времени: 2 с

Ограничение по памяти: 256 Мб

Однажды N программистов собрались на зимней школе по программированию в Харькове. Каждый из них написал код по супер-задаче. Затем главный тренер выстроил их в ряд и начал выяснять, у кого же код получился короче. После недолгих вычислений оказалось, что у k -го программиста длина кода равна a_k байт.

Тренер сразу же вычислил минимум, но на этом он не остановился. Для начала он придумал супер-число M , а затем он по очереди вызывал M подряд стоящих в ряду участников и находил среди них наикратчайший код. В итоге у него получилось $N - M + 1$ чисел.

А в качестве разминки Вам требуется повторить подвиг тренера.

Формат входного файла

Первая строка входного файла содержит два целых числа N и M ($1 \leq M \leq N \leq 3\,000\,000$) – количество участников школы и число, загаданное тренером. Во второй строке написаны N целых чисел a_k – длины кода каждого из участников, по модулю не превосходящие $1\,000\,000\,000$. Заметьте, что участники школы настолько суровы, что у них может получиться код нулевой длины, а у участников финала ACM ICPC даже и отрицательной!

Формат выходного файла

Выведите $N - M + 1$ целое число – эмуляцию ответов тренера.

Примеры

12 3	1
1 1 1 2 2 3 -1 8 15	1
10 3 1	1
	2
	-1
	-1
	-1
	8
	3
	1

День шестой (20.02.2014 г.) Конкурс Натальи Бондаренко

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 конкурса мне помогал мой товарищ по команде Дмитрий Матов.

Основные достижения:

- I диплом заключительного этапа Все-российской 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, VK Cup, Challenge 24, Russian Code Cup;
- 2 место в Russian Code Cup 2012;
- участница всех XIV сезонов Открытого Кубка имени Е.В. Панкратьева по программированию, в числе победителей во всех завершившихся сезонах, начиная с III-го.



Теоретический материал. Об одной комбинаторной игре

Задача (игра “цзяньшицы”, Wythoff’s game).

Имеются две кучки камней. Двою по очереди берут камни из этих кучек, причем при каждом ходе игрок может взять или произвольное количество камней из одной кучки, или поровну из обеих. Выигрывает тот, кто заберет последний камень. Определить, кто выиграет при правильной игре.

В задаче требуется установить, кто из игроков (первый или второй) имеет выигрышную стратегию, т.е. может выиграть при любых действиях другого игрока. Состояние игры определяется количествами камней в кучках a и b и может быть *выигрышным* или *проигрышным* относительно игрока, которому предстоит делать ход. Из состояния (a, b) существуют переходы в состояния $(a - k, b)$, $(a, b - k)$ и $(a - k, b - k)$ в соответствии с ходами игры. Получаем ациклический граф. Состояние будет выигрышным, если из него существует переход хотя бы в одно проигрышное состояние, и проигрышным, если все ходы из него ведут в выигрышные состояния. Пользуясь этим правилом, заполним таблицу:

$b \setminus a$	0	1	2	3	4	5	6	7
0		+	+	+	+	+	+	+
1	+	+	-	+	+	+	+	+
2	+	-	+	+	+	+	+	+
3	+	+	+	+	+	-	+	+
4	+	+	+	+	+	+	+	-
5	+	+	+	-	+	+	+	+
6	+	+	+	+	+	+	+	+
7	+	+	+	+	-	+	+	+

Динамика на ациклическом графе позволяет решить задачу за $O(n^3)$ операций для $a, b \leq n$. Хотелось бы получить более быстрое решение.

Заметим, что проигрышных состояний значительно меньше, чем выигрышных. В силу симметрии, нам достаточно рассмотреть состояния с $a < b$. Выпишем все такие проигрышные состояния (a_n, b_n) :

$$\begin{aligned} &(1, 2), \\ &(3, 5), \\ &(4, 7), \\ &(6, 10), \\ &(8, 13), \\ &(9, 15), \\ &(11, 18), \\ &(12, 20), \\ &(14, 23), \\ &\dots \end{aligned}$$

Хорошо прослеживаются две закономерности:

1. $b_n - a_n = n$, если позиции упорядочены по возрастанию a_n .
2. Каждое натуральное число встречается ровно в одной паре.

Свойства 1 и 2 однозначно определяют последовательность пар, которая может быть построена по следующему алгоритму за $O(n)$ операций. Будем помечать в массиве уже использованные числа. На каждом шаге берем первое неиспользованное число и прибавляем к нему n для получения второго числа пары. Будем называть пары, построенные по этому алгоритму, *особыми*.

Докажем, что все особые пары, и только они, соответствуют проигрышным состояниям.

А. Если начальная пара (a, b) ($a < b$) не является особой, то начинающий может выиграть одним ходом или свести игру к особой паре. Действительно, если числа a и b равны или одно из них равно нулю, то начинающий может выиграть одним ходом. Иначе найдем в списке особых пар число a . Пусть для определенности $a = a_n$ (случай $a = b_n$ рассматривается аналогично). Построим $b_n = a_n + n$. Если $b_n < b$, то возможен переход из (a, b) в (a_n, b_n) . В противном случае $b_n - a_n > b - a = m$. Тогда существует особая пара (a_m, b_m) с разностью m , причем $a_m < a_n$. Мы получаем ее из (a, b) вычитанием $a - a_m$ из обоих чисел.

Б. Если в начальном положении пара (a, b) является особой, то после любого хода начинающего она перестает быть особой. Действительно, если начинающий возьмет какое-то число камней из одной кучки, то пара перестанет быть особой благодаря свойству 2. Если он возьмет поровну камней из каждой кучки, то разность $b - a$ не изменится, а она не может быть одинаковой у двух особых пар в силу свойства 1.

Теперь исходное утверждение нетрудно доказать методом математической индукции по сумме $a + b$, пользуясь фактами А и В для перехода индукции.

Можно ли по паре (a, b) определить, особая она или нет, быстрее, чем за $O(n)$? Попытки найти в последовательности особых пар периодичность не дают результата. Для дальнейшего исследования задачи нам понадобится довольно нетривиальная теория.

Фibonacciева система счисления

Вспомним широко известную последовательность чисел Фибоначчи:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \dots$$

Эти числа определяются рекуррентным соотношением

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad n > 1.$$

Теорема Цекендорфа. *Любое неотрицательное целое число единственным образом представимо в виде суммы некоторого набора чисел Фибоначчи с индексами больше единицы, не содержащего пар соседних чисел Фибоначчи.*

В силу теоремы Цекендорфа, каждое неотрицательное целое число можно однозначно записать в виде последовательности нулей и единиц в *фибоначчиевой системе счисления*:

$$n = (b_m b_{m-1} \dots b_2)_F \Leftrightarrow n = \sum_{k=2}^m b_k F_k,$$

где $b_i = 1$, если F_i входит в представление, и 0 в противном случае.

Схема доказательства теоремы Цекендорфа.

Существование. Требуемое представление может быть получено для любого числа n при помощи жадного алгоритма. На каждом шаге выбираем наибольшее число Фибоначчи, не превосходящее n , вычитаем его из n и продолжаем процесс, пока $n > 0$.

Единственность. Предположим, для какого-то числа существует два представления. Возьмем наименьшее такое число. Тогда в одно его представление входит наибольшее число Фибоначчи F_m , а в другое — только числа с меньшими индексами. Нетрудно показать, что суммой чисел Фибоначчи с индексами, меньшими m и не идущими подряд, можно набрать только суммы, строго меньшие F_m .

Запишем в фибоначчиевой системе счисления найденные выше особые пары:

$$\begin{aligned} (1, 2) &= (1_F, 10_F), \\ (3, 5) &= (100_F, 1000_F), \\ (4, 7) &= (101_F, 1010_F), \\ (6, 10) &= (1001_F, 10010_F), \\ (8, 13) &= (10000_F, 100000_F), \\ (9, 15) &= (10001_F, 100010_F), \\ &\dots \end{aligned} \tag{1}$$

Здесь легко подмечается общая закономерность: во всех выписанных парах первое число оканчивается четным числом нулей (возможно, нулем), а второе получается из первого приписыванием в конце еще одного нуля (и, следовательно, оканчивается нечетным числом нулей). Докажем этот факт.

Для доказательства нам достаточно показать для имеющих такой вид в фибоначчиевой системе счисления пар выполнение свойств 1 и 2, которые однозначно определяют особые пары. Свойство 2 очевидно: числа с четным и

нечетным числом нулей на конце образуют разбиение множества натуральных чисел. Для доказательства свойства 1 запишем разность

$$\begin{aligned} b - a &= (F_{i_1+1} + F_{i_2+1} + \cdots + F_{i_m+1}) - (F_{i_1} + F_{i_2} + \cdots + F_{i_m}) = \\ &= F_{i_1-1} + F_{i_2-1} + \cdots + F_{i_m-1}. \quad (*) \end{aligned}$$

Если $i_1 > 2$, то $(*)$ представляет собой корректное число в фибоначиевой системе счисления с нечетным числом нулей на конце. Чуть сложнее случай $i_1 = 2$, потому что в этом случае в разложение $(*)$ входит F_1 . Тогда число a имеет вид $\dots 0010101\dots 01_F$. Поскольку $F_1 = F_2$, число $(*)$ имеет в фибоначиевой системе счисления представление $\dots 00101\dots 01011 = \dots 0100\dots 00_F$ с четным числом нулей на конце. Для разных чисел a будут получаться разные разности $b - a$. Наоборот, любое натуральное число n может быть однозначно представлено в виде $(*)$ и по нему можно однозначно построить исходное число a_n . Отсюда следует свойство 1.

Вывод: можно проверить, является ли заданная пара (a, b) особой, представив числа a и b в фибоначиевой системе счисления.

Спектры

Спектром вещественного числа $\alpha \geq 1$ называется множество

$$\text{Spec}(\alpha) = \{\lfloor \alpha \rfloor, \lfloor 2\alpha \rfloor, \lfloor 3\alpha \rfloor, \dots\}.$$

Здесь скобки $\lfloor . \rfloor$ обозначают целую часть числа. Например,

$$\text{Spec}(1.5) = \{1, 3, 4, 6, 7, 9, \dots\}.$$

Утверждение 1. $\text{Spec}(\alpha)$ имеет подобный периодический характер тогда и только тогда, когда число α рационально.

Применительно к игре “цзяньшицы”, нас интересуют разбиения множества натуральных чисел на две части.

Утверждение 2. Для того, чтобы спектры двух вещественных чисел α и β образовывали разбиение множества натуральных чисел:

$$\text{Spec}(\alpha) \cup \text{Spec}(\beta) = \mathbb{N}, \quad \text{Spec}(\alpha) \cap \text{Spec}(\beta) = \emptyset,$$

необходимо и достаточно, чтобы α и β были иррациональными числами, связанными соотношением

$$\frac{1}{\alpha} + \frac{1}{\beta} = 1.$$

Доказательство. *Достаточность.* Посчитаем суммарное количество чисел, меньших некоторого N , в двух спектрах. Оно равно

$$\left\lfloor \frac{N}{\alpha} \right\rfloor + \left\lfloor \frac{N}{\beta} \right\rfloor.$$

Так как числа $\frac{N}{\alpha}$ и $\frac{N}{\beta}$ иррациональны и

$$\frac{N}{\alpha} + \frac{N}{\beta} = N,$$

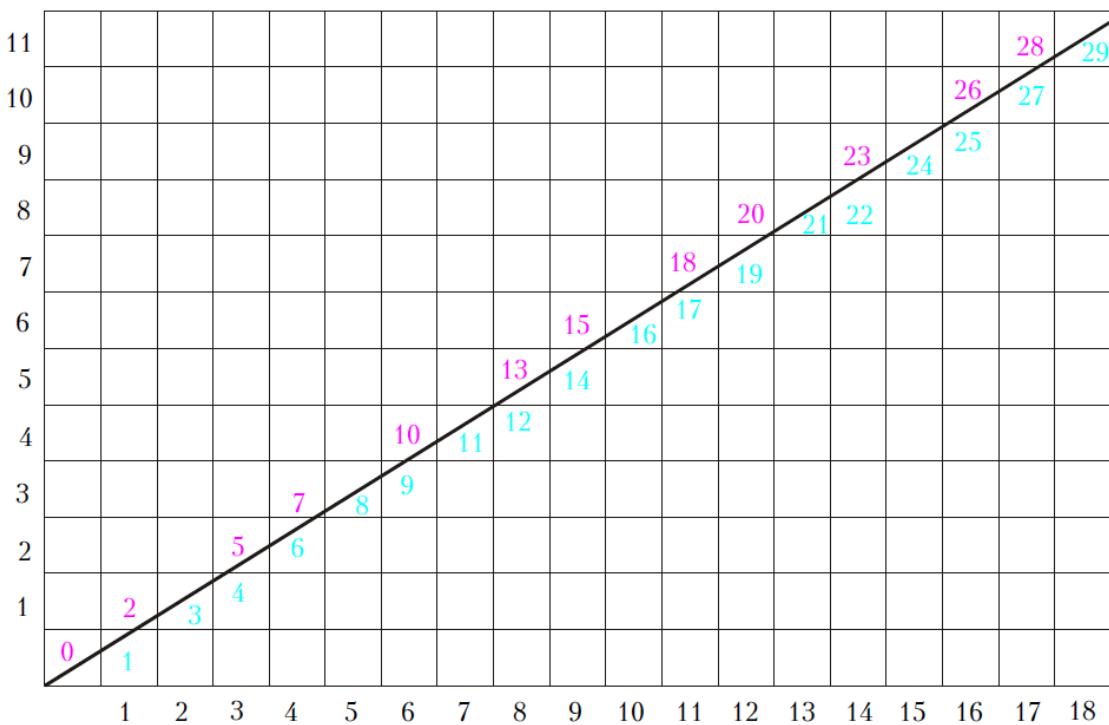
то

$$\left\lfloor \frac{N}{\alpha} \right\rfloor + \left\lfloor \frac{N}{\beta} \right\rfloor = N - 1.$$

Значит, существует одно число из спектров, меньшее 2 (число 1); два числа, меньшие 3 (1 и 2); три числа, меньшие четырех (1, 2, 3) и т.д. Каждое натуральное число встречается ровно в одном спектре.

Необходимость остается читателю в качестве упражнения. Тем более, в дальнейшем она нам не понадобится.

В [3] описано замечательное геометрическое доказательство достаточности в утверждении 2. Нарисуем на клетчатой бумаге как на координатной плоскости прямую, заданную уравнением $y = (\alpha - 1)x$, которое можно записать также в виде $x = (\beta - 1)y$. Занумеруем подряд все клетки, которые пересекает прямая. Поскольку число α иррационально, прямая не проходит через узлы сетки, кроме начала координат. Значит, прямая входит в очередную клетку либо слева, пересекая вертикальную линию сетки, либо снизу, пересекая горизонтальную линию.



Если прямая вошла в клетку слева и пересекла при этом вертикаль $x = n$, она ранее пересекла $n - 1$ вертикальную прямую и $\lfloor n(\alpha - 1) \rfloor$ горизонтальных. Поэтому номер клетки, в которую при этом вошла прямая, равен $n + \lfloor n(\alpha - 1) \rfloor = \lfloor n\alpha \rfloor$. Аналогично если прямая снизу пересекла горизонталь $y = m$, то номер соответствующей клетки равен $m + \lfloor m(\beta - 1) \rfloor = \lfloor m\beta \rfloor$. Клетки, пересеченные слева и пересеченные снизу, как раз образуют разбиение всего множества клеток, занумерованных натуральными числами.

Оказывается, можно подобрать такие числа α и β , чтобы их спектры образовывали последовательности $\{a_n\}$ и $\{b_n\}$. Эти числа должны удовлетворять условию утверждения 2 (тогда будет выполняться свойство 2 особых пар) и равенству $\lfloor n\beta \rfloor - \lfloor n\alpha \rfloor = n$ для выполнения свойства 1. Имеем

$$\lfloor n\beta \rfloor = \lfloor n\alpha \rfloor + n = \lfloor n(\alpha + 1) \rfloor \Rightarrow \text{Spec}(\beta) = \text{Spec}(\alpha + 1).$$

Из равенства спектров следует равенство самих чисел. Действительно, если $\alpha + 1$ и β различны, то при некотором достаточно большом n числа $n(\alpha + 1)$ и $n\beta$ будут отличаться больше чем на единицу и их целые части не совпадут. Получаем систему:

$$\frac{1}{\alpha} + \frac{1}{\beta} = 1, \quad \beta = \alpha + 1.$$

Из нее находим

$$\alpha^2 - \alpha - 1 = 0, \quad \alpha = \frac{1 + \sqrt{5}}{2}, \quad \beta = \frac{3 + \sqrt{5}}{2}.$$

Второй корень квадратного уравнения отрицательный, поэтому он нам не подходит. Число $\frac{1 + \sqrt{5}}{2}$ называется *золотым сечением*.

В итоге получаем

$$a_n = \left\lfloor n \frac{1 + \sqrt{5}}{2} \right\rfloor, \quad b_n = \left\lfloor n \frac{3 + \sqrt{5}}{2} \right\rfloor. \quad (2)$$

Эти формулы позволяют быстро строить n -е проигрышное состояние или определять по состоянию, выигрышное оно или проигрышное, за $O(1)$.

Связь чисел Фибоначчи и золотого сечения

Мы получили два эффективных решения задачи про “цзыньшицы” — через фибоначчиеву систему счисления и через спектры. Неужели формулы (1) и (2) как-то связаны между собой?

Для чисел Фибоначчи справедлива *формула Бине*:

$$F_n = \frac{\varphi_1^n - \varphi_2^n}{\varphi_1 - \varphi_2}, \quad \varphi_1 = \frac{1 + \sqrt{5}}{2}, \quad \varphi_2 = \frac{1 - \sqrt{5}}{2}.$$

Покажем, как она выводится. Рассмотрим линейную рекуррентность $v_n = v_{n-1} + v_{n-2}$. Ее *решением* называется любая последовательность $\{v_0, v_1, v_2, \dots\}$, удовлетворяющая данному соотношению при любом $n \geq 2$. Последовательность чисел Фибоначчи — одно из решений этой линейной рекуррентности. Будем искать степенные решения вида $v_n = \varphi^n$:

$$\varphi^n = \varphi^{n-1} + \varphi^{n-2} \Rightarrow \varphi — корень уравнения \varphi^2 - \varphi - 1 = 0.$$

Следовательно, линейная рекуррентность имеет два степенных решения $\{\varphi_1^n\}$ и $\{\varphi_2^n\}$. Любое другое решение представимо в виде их линейной комбинации: $v_n = a\varphi_1^n + b\varphi_2^n$. Действительно, любая последовательность такого вида дает решение, а любое решение однозначно определяется начальными условиями — значениями v_0 и v_1 . Мы всегда можем подобрать два коэффициента a и b , чтобы удовлетворить начальным условиям. Например, для чисел Фибоначчи имеем

$$\begin{aligned} a\varphi_1^0 + b\varphi_2^0 &= 0, \\ a\varphi_1^1 + b\varphi_2^1 &= 1, \end{aligned} \Rightarrow a = -b = \frac{1}{\varphi_1 - \varphi_2}.$$

Мы получили коэффициенты в формуле Бине. Аналогичным образом можно решать и другие линейные рекуррентности.

Рассмотрим теперь a_n в фибоначчиевой системе счисления и n в виде (*):

$$a_n = \sum_{k=1}^m F_{i_k} = \sum_{k=1}^m \frac{\varphi_1^{i_k} - \varphi_2^{i_k}}{\varphi_1 - \varphi_2},$$

$$n = \sum_{k=1}^m F_{i_k-1} = \sum_{k=1}^m \frac{\varphi_1^{i_k-1} - \varphi_2^{i_k-1}}{\varphi_1 - \varphi_2}.$$

Отсюда

$$n\varphi_1 - a_n = \sum_{k=1}^m \left(\frac{\varphi_1^{i_k} - \varphi_2^{i_k-1}\varphi_1}{\varphi_1 - \varphi_2} - \frac{\varphi_1^{i_k} - \varphi_2^{i_k}}{\varphi_1 - \varphi_2} \right) = - \sum_{k=1}^m \varphi_2^{i_k-1}.$$

Учитывая, что $-1 < \varphi_2 < 0$ и i_1 — четное, нетрудно показать, что полученная сумма дает положительное число меньше единицы. Так как a_n — целое, $\lfloor n\varphi_1 \rfloor = a_n$.

Список литературы

1. А.М. Яглом, И.М. Яглом. Неэлементарные задачи в элементарном изложении (1954), задача 129.
2. А. Матулис, А. Савукинас. Ферзя — в угол, “цзяньшицы” и числа Фибоначчи // “Квант” (1984).
3. А. Баабабов. “Пентиум” хорошо, а ум лучше // “Квант” (1999), геометрическое доказательство утверждения про спектры.
4. Р. Грэхем, Д. Кнут, О. Паташник. Конкретная математика. Основание информатики (1998), параграф 3.2: только про спектры, есть интересные упражнения.
5. Wikipedia.

Задачи и разборы

Задача А. Игра с числом

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

Петя и Вася придумали число n и играют с ним в игру. Ходят по очереди. Каждым ходом игрок делит имеющееся число на некоторый его делитель d , не являющийся полным квадратом, и заменяет n на результат от деления. Проигрывает тот, кто не может сделать ход. Первым ходит Петя.

Помогите ребятам определить, кто выиграет при оптимальной игре.

Формат входного файла Входные данные содержат одно целое число n ($1 \leq n \leq 10^9$).

Формат выходного файла Выведите номер игрока, который выигрывает при оптимальной игре.

Примеры

stdin	stdout
3	1
4	2

Примечание

В первом примере Петя делит число на 3 и выигрывает, потому что число 1 является полным квадратом и не имеет других делителей. Во втором примере Петя делает единственный возможный ход — делит число на 2 и получает 2. После этого Вася делит получившееся число на 2 и выигрывает.

Разбор задачи А. Игра с числом

Если число не является полным квадратом, то очевидно, что первый игрок может поделить его само на себя и за один ход выиграть. В противном случае после хода первого игрока останется не полный квадрат и после второго хода выиграет второй игрок.

Задача В. Отношение чисел Фибоначчи (простая версия)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 2 с
Ограничение по памяти: 512 Мб

У чисел Фибоначчи много замечательных свойств. В частности, отношение двух последовательных чисел Фибоначчи стремится к золотому сечению:

$$\lim_{n \rightarrow \infty} \frac{F_{n+1}}{F_n} = \frac{1 + \sqrt{5}}{2}.$$

Вам предстоит проверить это свойство для обобщенных чисел Фибоначчи

$$G_n = aG_{n-1} + bG_{n-2}, \quad n \geq 2.$$

По заданным a , b , G_0 и G_1 вычислите предел

$$\lim_{n \rightarrow \infty} \frac{G_{n+1}}{G_n},$$

в случае если он существует.

Формат входного файла

В первой строке заданы четыре целых числа a , b , G_0 , G_1 от 1 до 1000.

Формат выходного файла

В первой строке выведите “YES”, если предел существует, и “NO” в противном случае. В случае положительного ответа во второй строке выведите сам предел. Абсолютная или относительная погрешность ответа не должна превышать 10^{-6} .

Примеры

stdin	stdout
1 1 1 1	YES 1.6180339887

Разбор задачи В. Отношение чисел Фибоначчи (простая версия)

При ограничениях простой версии задачи предел всегда существует и может быть найден путем вычисления обобщенных чисел Фибоначчи и их отношения. Достаточно $n = 500$. Чтобы избежать переполнения, можно вычислять числа Фибоначчи в переменных вещественного типа (например, double) и нормировать, деля соседние числа на одинаковую константу.

Разберемся, почему это правильно. Характеристическое уравнение для заданной линейной рекуррентности имеет вид $\varphi^2 = a\varphi + b$. При ограничениях задачи оно всегда имеет два вещественных корня: положительный φ_+ и отрицательный φ_- , причем положительный больше по абсолютной величине. Число G_n представимо в виде $G_n = \alpha\varphi_+^n + \beta\varphi_-^n$. Коэффициент α не может быть нулевым, потому что тогда G_0 и G_1 имели бы разные знаки. Поэтому предел всегда равен большему корню:

$$\lim_{n \rightarrow \infty} \frac{G_{n+1}}{G_n} = \lim_{n \rightarrow \infty} \frac{\alpha\varphi_+^{n+1} + \beta\varphi_-^{n+1}}{\alpha\varphi_+^n + \beta\varphi_-^n} = \lim_{n \rightarrow \infty} \frac{\alpha\varphi_+(1 + \beta\frac{\varphi_-^{n+1}}{\varphi_+^{n+1}})}{\alpha(1 + \beta\frac{\varphi_-^n}{\varphi_+^n})} = \varphi_+.$$

Отличие отношения при фиксированном n от точного предела определяется величиной $\frac{\varphi_-}{\varphi_+}$. По теореме Виета $\varphi_+ + \varphi_- = a$, $\varphi_+\varphi_- = -b$, поэтому интересующая нас величина достигает максимума при наименьшем a и наибольшем b . Т.е. наихудший тест — $a = 1$, $b = 1000$. На нем можно экспериментально определить достаточное количество итераций. Другой вариант решения — найти корень φ_+ аналитически.

Задача С. Отношение чисел Фибоначчи (сложная версия)

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

У чисел Фибоначчи много замечательных свойств. В частности, отношение двух последовательных чисел Фибоначчи стремится к золотому сечению:

$$\lim_{n \rightarrow \infty} \frac{F_{n+1}}{F_n} = \frac{1 + \sqrt{5}}{2}.$$

Вам предстоит проверить это свойство для обобщенных чисел Фибоначчи

$$G_n = aG_{n-1} + bG_{n-2}, \quad n \geq 2.$$

По заданным a , b , G_0 и G_1 вычислите предел

$$\lim_{n \rightarrow \infty} \frac{G_{n+1}}{G_n},$$

в случае если он существует.

Формат входного файла

В первой строке заданы четыре целых числа a , b , G_0 , G_1 от -1000 до 1000 .

Формат выходного файла

В первой строке выведите “YES”, если предел существует, и “NO” в противном случае. В случае положительного ответа во второй строке выведите сам предел. Абсолютная или относительная погрешность ответа не должна превышать 10^{-6} .

Примеры

stdin	stdout
1 1 1 1	YES 1.6180339887
0 0 1 1	NO

Примечание

Если последовательность G_n содержит бесконечное количество нулей, считается, что предела не существует.

Разбор задачи С. Отношение чисел Фибоначчи (сложная версия)

Рекомендуется сначала прочитать решение простой версии задачи.

Алгоритм решения, описанный в простой версии, в данном случае не работает. Во-первых, характеристическое уравнение может иметь только один корень φ_1 . Тогда линейная рекуррентность имеет фундаментальную систему решений $\{\varphi_1^n\}$, $\{n\varphi_1^n\}$. Искомое отношение будет стремиться к корню φ_1 , но гораздо медленнее, необходимо $\approx 10^6$ итераций. Во-вторых, возможен случай $G_n = \beta\varphi_-^n$, т.е. в разложении участвует только меньший по модулю корень. В этом случае численное решение неустойчиво. Из-за погрешности вычисляемая последовательность отношений будет сходиться не к меньшему, а к большему корню (проверьте!).

Возможно аналитическое решение путем нахождения корней характеристического уравнения. Нужно отдельно исследовать случай с комплексными корнями (можно показать, что в этом случае предела не существует), случай с двумя одинаковыми по абсолютной величине корнями разного знака ($a = 0$) и случай $b = 0$.

Другой вариант решения — построить последовательность

$$x_n = \frac{G_{n+1}}{G_n} = \frac{aG_n + bG_{n-1}}{G_n} = a + \frac{bG_{n-1}}{G_n} = a + \frac{b}{x_{n-1}}, \quad x_0 = \frac{G_1}{G_0}$$

и проверить ее на сходимость.

В обоих предложенных решениях нужно отдельно рассматривать случаи с нулями в последовательности G_n , чтобы избежать деления на ноль.

Задача D. Обобщенная последовательность Фибоначчи

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

Назовем обобщенной последовательностью Фибоначчи последовательность, первые два элемента которой равны a и b (a и b — положительные целые числа), а каждый следующий элемент равен сумме двух предыдущих.

Таким образом, каждая такая последовательность однозначно определяется её первыми двумя членами. Например, если $a = 3, b = 4$, то первые члены последовательности: $3, 4, 7, 11, 18, 29, \dots$. Стандартная последовательность чисел Фибоначчи получается при $a = b = 1$.

Вам дано число n . Требуется найти количество обобщенных последовательностей Фибоначчи, в которых n встречается в качестве i -ого элемента, где $i > 2$. Элементы обобщенной последовательности нумеруются с единицы.

Формат входного файла

Входные данные содержат единственное число n ($1 \leq n \leq 10^6$).

Формат выходного файла

Выведите единственное число — ответ на задачу.

Примеры

stdin	stdout
3	3
1	0

Разбор задачи D. Обобщенная последовательность Фибоначчи

Пусть $F_i = n$, переберем F_{i-1} от 1 до $n - 1$. По двум членам обобщенной последовательности Фибоначчи можно достроить все предыдущие по формуле $F_{k-2} = F_k - F_{k-1}$. Так можно получить все пары положительных чисел (a, b) , с которых могла начинаться последовательность. Поскольку $i = O(\log F_i)$, общее время работы алгоритма $O(n \log n)$.

Другое решение: заметим, что $F_i = af_{i-2} + bf_{i-1}$, где f_i — стандартные числа Фибоначчи ($f_1 = f_2 = 1$). Поэтому можно перебрать все подходящие значения i и перебрать все a такие, что $af_{i-2} \leq n$, и проверить, что $b = \frac{F_i - af_{i-2}}{f_{i-1}}$ — целое число. Нетрудно показать, что количество пар (i, a) , которые потребуется проверить, не превосходит $O(n \log n)$.

Задача Е. Сложение в фибоначчиевой системе счисления

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 2 с
Ограничение по памяти: 512 Мб

Числа Фибоначчи определяются рекуррентным соотношением:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \text{ для } n > 1.$$

Фибоначчиева система счисления (ФСС) основана на теореме Цекендорфа, утверждающей, что любое целое положительное число имеет единственное представление вида

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_r},$$

где F_{k_i} — числа Фибоначчи, а $k_1 \gg k_2 \gg \cdots \gg k_r \gg 0$. Здесь $i \gg j$ означает, что $i \geq j + 2$. Целое неотрицательное число можно записать в виде последовательности нулей и единиц:

$$n = (b_m b_{m-1} \dots b_2)_F \Leftrightarrow n = \sum_{k=2}^m b_k F_k,$$

где $b_i = 1$, если F_i входит в представление, и 0 в противном случае. Например, $1000000 = 832040 + 121393 + 46368 + 144 + 55 = F_{30} + F_{26} + F_{24} + F_{12} + F_{10}$, или $(1000000)_10 = (10001010000000001010000000)_F$. Эта система счисления напоминает двоичное представление, за исключением того, что в ней никогда не встречаются две единицы подряд.

При прибавлении единицы к числу в ФСС возникают два случая. Если младший разряд есть нуль, он заменяется на единицу (так как $F_2 = 1$), в противном случае в двух младших разрядах будет 01, и они заменяются на 10 (так как $F_3 = F_2 + 1$). Затем мы должны заменять набор цифр 011 на 100 (так как $F_n = F_{n-1} + F_{n-2}$), до тех пор, пока в строке цифр имеются две рядом стоящие единицы.

Напишите программу для сложения двух чисел в ФСС.

Формат входного файла

Вам даны два неотрицательных целых числа в ФСС, по одному числу в строке. Количество разрядов в числах может быть различным и не превосходит 5000.

Формат выходного файла

Выведите результат сложения этих двух чисел также в ФСС.

Пример

stdin	stdout
1010	10010
100	

Разбор задачи Е. Сложение в фибоначчиевой системе счисления

Первое решение — реализовать алгоритм сложения в ФСС, пользуясь таблицей сложения:

$$\begin{array}{rcl} 0 + 0 = & 0 \\ 0 + 1 = & 1 \\ 0 + 1 = & 1 \\ 1 + 1 = & 1 \ 1 \ 1 \quad (a) \\ 1 + 1 = & 1 \ 0 \ 0 \ 1 \quad (b) \end{array}$$

Отличительная особенность этой таблицы — перенос не только в более старшие, но и в более младшие разряды. При сложении двух единиц в k -й позиции (т.е. $F_k + F_k$) можно представить одно из чисел F_k в виде суммы $F_{k-1} + F_{k-2}$ и получить правило (а). После этого сгруппируем $F_k + F_{k-1} = F_{k+1}$ и получим из правила (а) правило (б). Чтобы корректно обработать перенос в младшие разряды, будем действовать по алгоритму:

1. Перебрать разряды второго числа от старших к младшим. Если в текущем разряде стоит единица, прибавить ее к соответствующему разряду первого числа (шаг 2).
2. Осуществить прибавление единицы в k -м разряде в соответствии с таблицей, в случае двух единиц используя правило (б). Если образовались две подряд стоящие единицы в позициях k и $k + 1$ или $k + 1$ и $k + 2$, произвести нормализацию (шаг 3). Затем, если применялось правило (б), выполнить шаг 2 для прибавления единицы в $k - 2$ разряде.
3. Нормализация: заменить комбинацию 011 на 100. В случае образования новой комбинации 011 на две позиции левее, продолжить процесс замен до остановки.

Оценим время работы алгоритма. Шаг 2 будет выполняться $O(n^2)$ раз, где n — длина чисел. Количество операций при нормализации не будет превышать $O(n)$ плюс количество выполнений шага 2. Действительно, при нормализации будет происходить много действий, если имеется длинная последовательность $\dots 10101010$. На каждом шаге нормализации уничтожается одна

пара 10, а новые пары 10 появляются только при выполнении шага 2. Поэтому общее время работы алгоритма — $O(n^2)$.

Второе решение — перевести оба числа в десятичную систему счисления, сложить их при помощи длинной арифметики и перевести результат обратно в ФСС, используя жадный алгоритм, рассказанный на лекции.

Задача F. Игра с монетами

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

Петя и Вася придумали игру с монетами. У них есть n монет достоинствами a_i . В наборе могут быть монеты одинаковых достоинств. Сначала все монеты лежат на столе. Игроки по очереди берут по одной монете со стола. Первым ходит Петя. Он выигрывает, если после очередного хода у него окажется наперед заданная и известная обоим игрокам сумма s . Возможно, в этот момент на столе еще будут оставаться другие монеты. Вася выигрывает, если ему удастся помешать Пете набрать заданную сумму.

Петя считает игру нечестной, потому что сумм s , при которых он может выиграть, достаточно мало. Помогите ребятам определить все такие суммы s , которые Петя может набрать независимо от действий Васи.

Формат входного файла

В первой строке содержится целое число n ($1 \leq n \leq 100$) — количество монет на столе. В следующей строке заданы n целых положительных чисел — достоинства монет a_i . Их общая сумма не превосходит 100.

Формат выходного файла

В первой строке выведите количество сумм s , выигрышных для Пети. В следующей строке через пробел выведите сами суммы в порядке возрастания.

Примеры

stdin	stdout
3	3
2 2 3	2 3 5

Разбор задачи F. Игра с монетами

Состояние игры характеризуется набором монет, оставшихся на столе, суммой, которую Пете осталось набрать, и номером игрока, который ходит. За счет ограничения на общую стоимость монет, общее количество возможных наборов не превосходит 300 000 (если считать, что монеты одного достоинства неразличимы). Поэтому задачу можно решать динамикой по состояниям игры.

Задача G. Цзяньшицы с тремя кучками

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

Имеются три кучки камней. Двое по очереди берут камни из этих кучек, причем при каждом ходе игрок может взять или произвольное количество камней из одной кучки, или поровну из всех трех. Выигрывает тот, кто заберет последний камень. Определите, кто выиграет при оптимальной игре.

Формат входного файла

Входные данные содержат несколько тестов. В первой строке содержится их количество t ($1 \leq t \leq 1000$). Следующие t строк описывают сами тесты. Каждая из них содержит тройку чисел a, b, c ($1 \leq a, b, c \leq 10^{12}$) — количества камней в кучках.

Формат выходного файла

Для каждого теста в отдельной строке выведите номер игрока (1 или 2), который выиграет при правильной игре. В случае если выиграет первый игрок, в той же строке через пробел выведите первый ход, который приведет его к победе. Ход описывается двумя целыми числами: номером кучки и строго положительным количеством камней, которые нужно из нее взять. Кучки нумеруются числами 1, 2, 3. Если предполагается взять поровну камней из всех трех кучек, в качестве номера кучки выведите 0. Если вариантов выигрышного хода несколько, выведите любой. Если выигрывает второй игрок, ход выводить не нужно.

Примеры

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

Разбор задачи G. Цзяньшицы с тремя кучками

В случае трех кучек игра «цзяньшицы» эквивалентна ниму, т.е. игре без возможности взятия равного количества камней из трех кучек одновременно. Рассмотрим ациклический граф выигрышных и проигрышных состояний для нима с тремя кучками. Добавление нового вида хода влечет за собой добавление новых дуг в этот граф. Покажем, что не появится дуг из проигрышных состояний в выигрышные. Тогда выигрышность и проигрышность состояний при переходе к игре «цзяньшицы» сохранится.

Действительно, числа $a \oplus b \oplus c$ и $(a - k) \oplus (b - k) \oplus (c - k)$ (\oplus — побитовый XOR) не могут быть одновременно равны нулю, поскольку они отличаются в позиции младшего единичного бита числа k . Значит, состояния (a, b, c) и $(a - k, b - k, c - k)$ не могут оба быть проигрышными.

То же самое справедливо для игры с любым нечетным числом кучек. «Цзяньшицы» с четным числом кучек — значительно более сложная задача.

Задача H. Спички детям не игрушки

Вход:	stdin
Выход:	stdout
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

Петя и Вася снова играют в игру со спичками. У них имеется куча спичек, из которой игроки по очереди берут спички по определенным правилам. Проигрывает тот, кто не может сделать ход.

Варианты игры, в которых за ход разрешается брать не более фиксированного количества спичек, извечным персонажам олимпиадных задач уже давно надоели. Поэтому сегодня они договорились, что количество спичек, которое игрок может брать за ход — от одной до $\lfloor \log_2 n \rfloor$ (округленного вниз двоичного логарифма числа спичек n перед текущим ходом).

Несмотря на то, что игра обещает быть интересной, у Пети с Васей нет возможности ее доиграть. С минуты на минуту придет бабушка, которая игры

со спичками категорически не одобряет. Помогите ребятам определить, кто из них выиграет при оптимальной игре.

Формат входного файла

Входные данные содержат несколько тестов. В первой строке содержится их количество t ($1 \leq t \leq 1000$). Следующие t строк описывают сами тесты. Каждая из них содержит целое число n ($1 \leq n \leq 10^{18}$) — количество спичек в кучке в начале игры.

Формат выходного файла

Для каждого теста в отдельной строке выведите номер игрока (1 или 2), который выигрывает при правильной игре. В случае если выигрывает первый игрок, в той же строке через пробел выведите первый ход, который приведет его к победе — число спичек, которое нужно взять из кучки. Если вариантов выигрышного хода несколько, выведите любой. Если выигрывает второй игрок, ход выводить не нужно.

Примеры

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

Примечание

$\log_2 1 = 0$, поэтому если в кучке останется одна спичка, игра заканчивается.

Разбор задачи Н. Спички детям не игрушки

Нетрудно получить ответ для первых значений n и заметить закономерность: при n в интервале 2^k до $2^{k+1} - 1$ проигрышные состояния идут через каждые k выигрышных. Это происходит потому, что максимальное количество спичек за ход для таких n одинаково и равно k . Используя этот факт, можно для каждой степени двойки посчитать самое близкое меньшее ее проигрышное состояние и затем по этим состояниям вычислять ответы на запросы.

Задача I. Ферзя в угол!

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 2 с
Ограничение по памяти: 512 Мб

В левом верхнем углу шахматной доски $n \times m$ стоит ферзь. Двое играют в следующую игру. Ходят по очереди. Каждым ходом разрешается передвинуть ферзя по вертикали вниз, по горизонтали вправо или по диагонали вправо-вниз на любое ненулевое число клеток. Проигрывает тот, после чьего хода ферзь окажется в правом нижнем углу доски.

Как вы уже догадались, ваша задача определить, кто выиграет при правильной игре.

Формат входного файла

Входные данные содержат несколько тестов. В первой строке содержится их количество t ($1 \leq t \leq 1000$). Следующие t строк описывают сами тесты. Каждая из них содержит пару чисел n и m ($2 \leq n, m \leq 10^{12}$), n — высота доски, m — ширина доски.

Формат выходного файла

Для каждого теста в отдельной строке выведите номер игрока (1 или 2), который выиграет при правильной игре. В случае если выиграет первый игрок, в той же строке через пробел выведите первый ход, который приведет его к победе. Ход описывается двумя целыми неотрицательными числами — смещением по вертикали a и смещением по горизонтали b . Заметим, что у корректного хода ферзя $a = 0$, или $b = 0$, или $a = b$, при этом a и b не могут быть нулями одновременно. Если вариантов выигрышного хода несколько, выведите любой. Если выиграет второй игрок, ход выводить не нужно.

Пример

stdin	stdout
3	2
3 3	1 1 1
5 7	1 9 0
10 2	

Разбор задачи I. Ферзя в угол!

Эта игра эквивалентна игре “цзяньшицы” с кучками размеров $a = n - 1$ и $b = m - 1$ и тем отличием, что проигрывает игрок, делающий последний

ход. Проигрышные состояния (a, b) получаются такими же, как для классического варианта игры, за исключением нескольких первых. А именно, вместо $(0, 0)$ и $(1, 2)$ проигрышными теперь будут состояния $(0, 1)$ и $(2, 2)$. Поэтому к задаче применимо решение с разложением в фибоначчиевой системе счисления, описанное на лекции, с отдельным рассмотрением некоторых частных случаев. Решение с золотым сечением при ограничениях задачи не проходит по точности.

Задача J. (p, q) -коня в угол!

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

(p, q) -конь — это фигура, которая за один ход перемещается на p клеток в одном направлении (по вертикали или по горизонтали) и на q клеток в перпендикулярном ему. Для обычного шахматного коня $p = 1, q = 2$.

(p, q) -конь стоит в левом верхнем углу $(1, 1)$ шахматной доски $n \times m$ и ему нужно добраться до правого нижнего угла (n, m) . В этой задаче конь может совершать только такие ходы, которые приближают его к цели и в горизонтальном, и в вертикальном направлении. То есть он может пойти либо на p клеток вправо и на q клеток вниз, либо наоборот: на q клеток вправо и на p вниз. Разумеется, конь не может выходить за пределы доски. Кроме того, на доске есть запрещенные клетки, по которым также нельзя ходить (но через которые можно перепрыгивать). Клетки $(1, 1)$ и (n, m) не могут быть запрещенными.

Определите количество способов добраться из клетки $(1, 1)$ в клетку (n, m) , не посещая запрещенные клетки. Поскольку это количество может быть очень большим, выведите его по данному простому модулю d .

Формат входного файла

В первой строке содержатся параметры коня p и q и простой модуль d ($1 \leq p < q \leq 20, 2 \leq d \leq 10^6$). Во второй строке заданы размеры доски n и m ($2 \leq n, m \leq 10^9$). В следующей строке содержится одно число k — количество запрещенных клеток ($0 \leq k \leq 10$). В следующих k строках перечислены запрещенные клетки парами координат “ $x_i y_i$ ” ($1 \leq x_i \leq n, 1 \leq y_i \leq m$). Все запрещенные клетки различны между собой и не совпадают с начальной и конечной клетками $(1, 1)$ и (n, m) .

Формат выходного файла

Выведите одно целое число — ответ на задачу по модулю d .

Примеры

stdin	stdout
1 2 23 4 4 0	2
1 2 23 4 4 1 3 2	1

Разбор задачи J. (p, q) -коня в угол!

1. **Формула включений-исключений.** Сразу обратим внимание на то, что размеры поля довольно большие, но запрещенных клеток мало — не больше 10. Переберем все подмножества S запрещенных клеток и для каждого подмножества посчитаем количество путей (p, q) -коня из клетки $(1, 1)$ в клетку (n, m) , проходящих через все клетки S и возможно через какие-то другие запрещенные клетки. Обозначим эту величину $A(S)$. Тогда по формуле включений-исключений ответ на задачу равен

$$\sum_S (-1)^{|S|} A(S), \quad |S| — \text{мощность множества } S.$$

2. **Вычисление $A(S)$.** Ясно, что при вычислении $A(S)$ запрещенные клетки из S проходятся в порядке возрастания обеих координат: $1 < x_1 < x_2 < \dots < x_r < n$, $1 < y_1 < y_2 < \dots < y_r < m$. Если клетки нельзя упорядочить таким образом, $A(S) = 0$. Иначе $A(S)$ равно произведению количеств путей между парами соседних клеток в последовательности.
3. **Количество путей между парой клеток.** Обозначим $s = x_2 - x_1$, $t = y_2 - y_1$, x — количество ходов (p, q) и y — количество ходов (q, p) для перехода из (x_1, y_1) в (x_2, y_2) . Тогда x и y могут быть однозначно найдены из системы

$$\begin{cases} px + qy = s, \\ qx + py = t, \end{cases}$$

если система разрешима в целых неотрицательных числах. В противном случае количество путей равно нулю. Пути отличаются относитель-

ным порядком ходов вида (p, q) и (q, p) . Поэтому количество путей равно количеству сочетаний $C_{x+y}^x = \frac{(x+y)!}{x!y!}$.

4. **Теорема Люка** дает удобный метод вычисления биномиальных коэффициентов по произвольному простому модулю. Пусть

$$n = n_0 + n_1p + n_2p^2 + \cdots + n_mp^m,$$
$$k = k_0 + k_1p + k_2p^2 + \cdots + k_mp^m$$

— представления чисел n и k в p -ичной системе счисления. Тогда

$$C_n^k = \prod_{i=0}^m C_{n_i}^{k_i} \pmod{p}.$$

Коэффициенты $C_{n_i}^{k_i}$ ($n_i, k_i < p$) можно вычислить, предварительно вычислив факториалы и обратные к ним величины для всех чисел, меньших модуля.

Задача K. Игра

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

В аристократических кругах Берляндии получила распространение следующая игра. Поле для этой игры представляет собой l ячеек, расположенных по кругу. В некоторых ячейках расположены фишками белого и черного цвета, причем в одной ячейке не может быть более одной фишкой. Фишками белого цвета играет первый игрок, фишками черного цвета — второй. Игроки делают ходы по очереди. Первым ходит игрок, фишками которого белые. За один ход игрок может взять любую фишку своего цвета и подвинуть ее на ненулевое количество клеток по или против часовой стрелки. При этом при перемещении не разрешается перепрыгивать ни через какие другие фишки или занимать ячейку, в которой уже находится какая-то фишка. Проигрывает в игре тот игрок, кто не может сделать очередной ход. Учитывая, что оба противника играют оптимально, определите исход игры. Игра может завершиться ничьей (т.е. может продолжаться бесконечно).

Формат входного файла

Входной файл состоит из нескольких тестовых примеров. В первой строке записано целое число t , обозначающее количество этих приме-

ров. Описание каждого теста начинается с тройки целых чисел l , n и m ($1 \leq l \leq 10^9$, $1 \leq n, m \leq 10^6$) — количество ячеек, количество белых фишек и количество черных фишек соответственно. Далее следует возрастающая последовательность из n целых чисел a_1, a_2, \dots, a_n ($1 \leq a_i \leq l$) — номера ячеек, в которых стоят белые фишкы. Затем следует возрастающая последовательность из m целых чисел b_1, b_2, \dots, b_m ($1 \leq b_i \leq l$) — номера ячеек, в которых стоят черные фишкы. Ячейки пронумерованы от 1 до l последовательно по часовой стрелке. Все числа a_i и b_j различны. Суммы значений n и m по всем тестовым примерам не превосходят 10^6 .

Формат выходного файла

Для каждого тестового примера выведите ответ на отдельной строке. Если победит игрок,двигающий белые фишкы, выведите “white”. В случае, если победит игрок,двигающий черные фишкы, выведите “black”. Если же игра завершится ничьей, выведите “draw”.

Примеры

stdin	stdout
4	black
2 1 1	draw
1	white
2	white
4 1 1	
1	
2	
4 2 1	
1 4	
2	
6 2 2	
5 6	
2 4	

Разбор задачи К. Игра

Рассмотрим все возможные случаи расположения фишек.

- На поле нет свободных позиций, поэтому белые не могут сделать ни одного хода. В этом случае выигрывают черные.

Далее на рисунках точки обозначают, что между фишками есть одна или несколько свободных позиций.

1. Ни у кого из игроков нет группы более чем из одной фишки (1A). В этой ситуации никто не может выиграть, “зажав” фишку другого игрока (1B).

Рисунок 1А

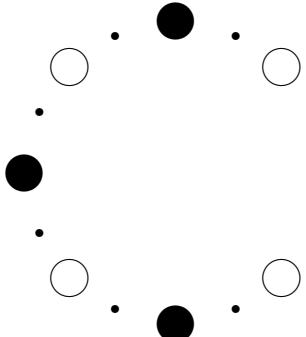


Рисунок 1В

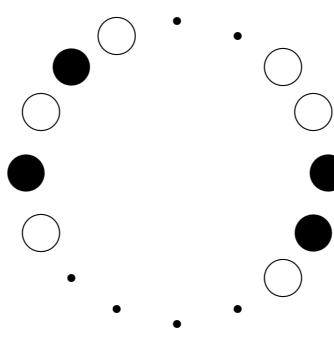
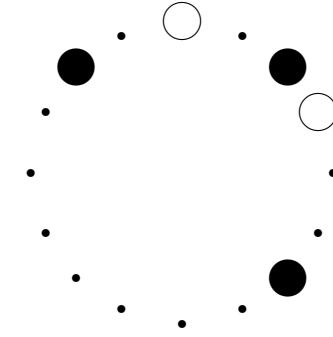


Рисунок 2А



2. У одного игрока есть пара стоящих подряд фишек, а у второго игрока — нет (2А). Можно показать, что в этом случае выигрывает игрок, имеющий пару.
3. На поле стоят две черные и две белые фишки, как показано на рисунке 3А. Покажем, что в этом случае игра эквивалентна ниму с кучками l_1 и l_2 (количество свободных позиций между фишками в парах “черная-белая”). Будем называть ходами «вперед» ходы в направлении к фишке противоположного цвета. Игрок, который имеет выигрышную стратегию в ниме, должен ходить вперед, уменьшая длины l_1 и l_2 в соответствии с этой стратегией. Другой игрок может ходить вперед (тогда на его ходы можно отвечать в соответствии с нимом) или назад (тогда сопернику нужно ходить вперед на такое же количество позиций). Из-за ходов назад расстояние между двумя фишками проигрывающего сокращается, поэтому в конце концов другой игрок его “запрет” (3В).

Приведенные рассуждения обобщаются на случай нескольких чередующихся пар черных и белых фишек. Например, случай на рисунке 3С эквивалентен ниму с кучками l_1, l_2, l_3, l_4 , т.е. ответ зависит от величины

$$l_1 \oplus l_2 \oplus l_3 \oplus l_4.$$

Рисунок 3А

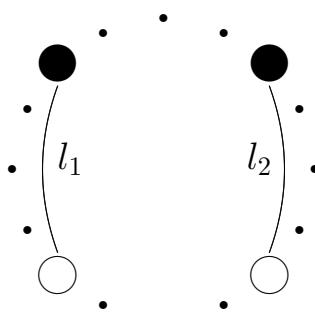


Рисунок 3В

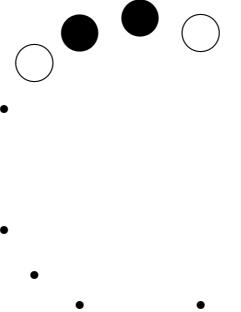
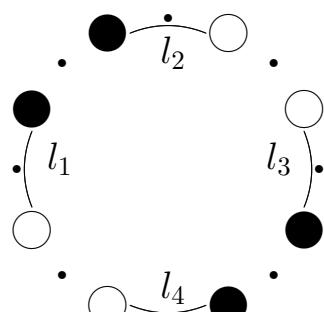


Рисунок 3С



4. Ситуация 4А аналогична предыдущей: белые фишки наступают на черные, а черные — на белые. Игра сводится к ниму с кучками l_1 , l_2 и l_3 . Аналогично решаются подобные ситуации с большим количеством фишек.

Рисунок 4А

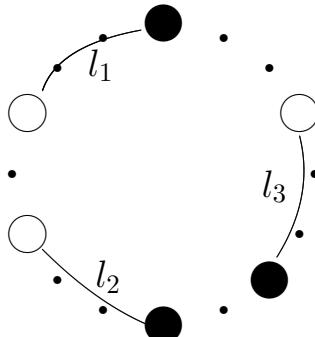


Рисунок 5А

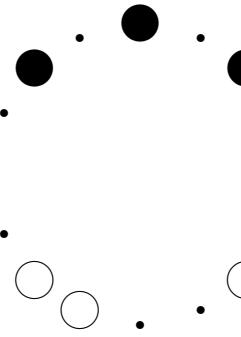
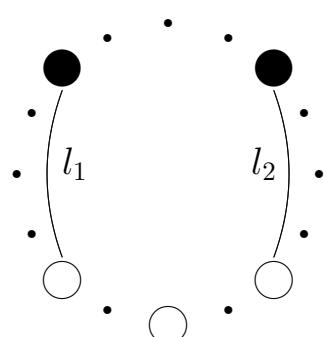


Рисунок 5В



5. **“Открытая тройка”**. Если у игрока имеются три подряд идущие фишки (5А), между которыми есть хотя бы одна свободная позиция, то этот игрок не может проиграть. Действительно, он всегда может сделать ход фишкой, стоящей в середине. Если у обоих игроков есть открытые тройки — будет ничья (5А). В противном случае выиграет игрок, имеющий тройку. Например, в позиции на рисунке 5В белые могут играть в ниме выигрышный, и ходить третьей фишкой в противном случае, передавая проигрышную позицию черным.

6. **“Открывающаяся тройка”**. Нетрудно показать, что закрытая тройка

(внутри которой нет свободных позиций) ведет себя как пара (6A).

Рисунок 6A

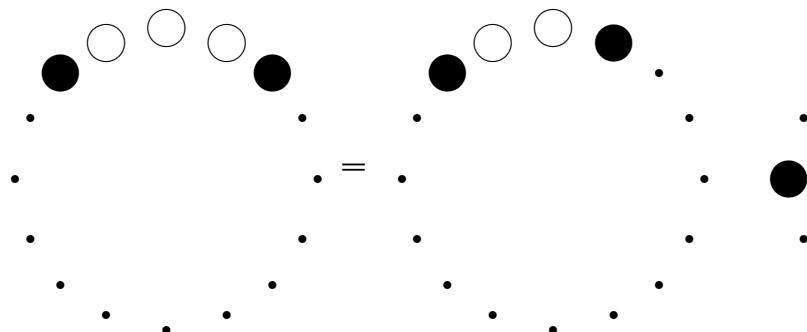
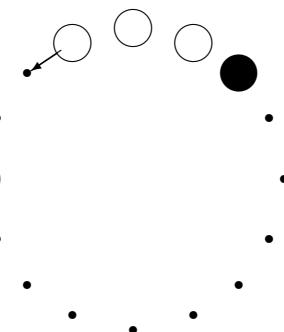


Рисунок 6B



Однако закрытую тройку иногда можно открыть. Если, например, она у белых и рядом есть свободная позиция (6B). А если открывающаяся тройка у черных? Тогда белые могут закрыть ее первым же ходом, а могут использовать этот ход по-другому (например, для открытия своей тройки) — рисунок 6C. Этот случай удобно разобрать рекурсивным перебором, моделируя поведение игроков до тех пор, пока либо не появится открытая тройка, либо не исчезнут открывающиеся тройки. Одно из двух произойдет быстро, поскольку в случае наличия двух открывающихся троек у одного игрока он обязательно сможет открыть одну из них (6D).

Рисунок 6C

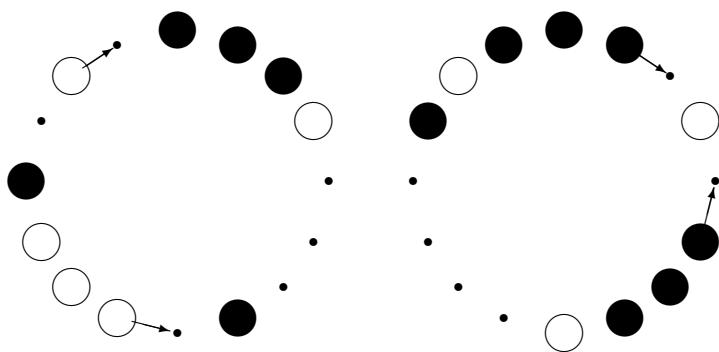
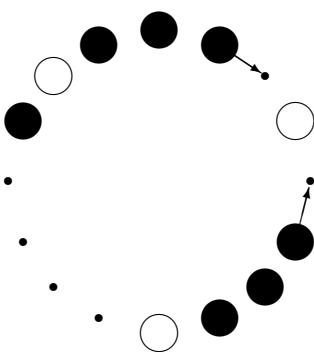


Рисунок 6D



7. Итак, мы разобрали все случаи с тройками и свели задачу к позиции без троек. Еще не разобран случай 7A, с последовательностью нечетной длины из чередующихся фишек. Нетрудно показать, что такая последова-

тельность ведет себя как тройка: открытая (7A) или закрытая (7B).

Рисунок 7А

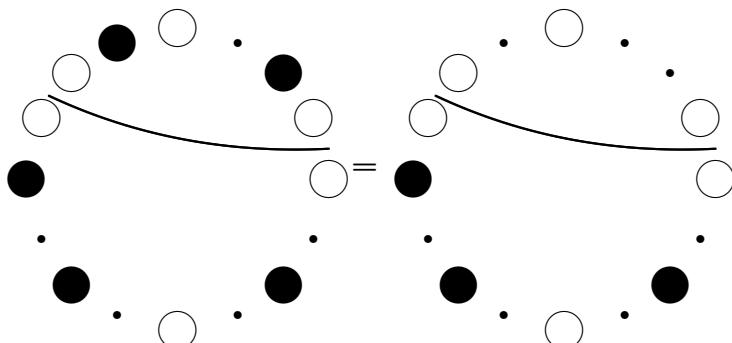
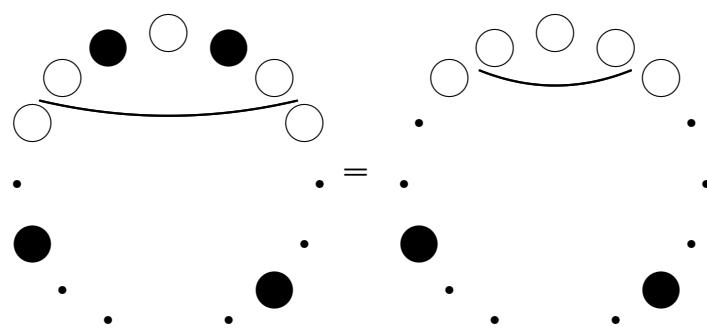


Рисунок 7В



Подведем итоги. Разбор случаев привел к следующему алгоритму решения задачи.

1. Обработаем случаи 0–2.
2. Избавимся от нечетных последовательностей единиц, заменив их на тройки (случай 7).
3. Если есть тройки, разберем случаи с тройками 5–6.
4. После избавления от троек мы пришли к случаю 3 или 4. Считаем функцию Гранди для соответствующего нима.

Задача L. Игра со строкой

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

Это интерактивная задача. Ваша программа должна читать данные из стандартного входного потока и выводить данные в стандартный выходной поток. После вывода каждой строки используйте flush-операции, например, `fflush(stdout)` в C++, `System.out.flush()` в Java, `flush(output)` в Pascal, `sys.stdout.flush()` в Python, `Console.Out.Flush()` в C#.

Представьте, что на доске написана строка и двое играют с ней в следующую игру. Игроки делают ходы по очереди. Ход состоит в стирании некоторого непустого подмножества символов строки. При этом не разрешается стирать одним ходом два одинаковых символа. Например, из строки “`abacaba`” можно стереть символы с номерами 1, 2 и 4, но нельзя стереть символы с номерами 1, 3, 5, 6, потому что, например, первый и третий символы — одинаковые (“`a`”). Проигрывает тот, кто не может сделать ход, т.е. перед чьим ходом на доске не остается ни одного символа.

Ваша задача состоит в том, чтобы выбрать, за какого игрока играть (за первого или за второго) и обыграть программу жюри (интерактор) в эту игру.

Формат входного файла

Входные данные содержат непустую строку s длины до 500, состоящую из строчных латинских букв. Символы строки нумеруются целыми числами от 1 до ее длины. При стирании символов номера остальных символов *не меняются*.

При обмене информацией с интерактором на вход подаются ходы оппонента, по одному в строке. Описание хода содержит целое положительное число — количество элементов в стираемом подмножестве. Далее в той же строке должны быть перечислены номера стираемых символов в любом порядке. Числа в описании хода разделяются пробелами. Гарантируется, что ходы интерактора корректны — все стираемые символы различны, каждый номер выводится только один раз и не совпадает с номером никакого уже стертого ранее символа. Ваши ходы тоже должны удовлетворять этим условиям.

Если вы решаете играть за первого игрока, выведите свой первый ход, дальше прочтайте ход второго игрока, потом опять выведите свой ход и т.д. Если вы играете за второго игрока, сначала прочтайте ход оппонента, потом выведите свой ход, потом снова прочтайте ход оппонента и т.д.

Когда оппонент проигрывает, он вместо своего хода выводит одно число “ -1 ”, после считывания которого ваша программа должна завершить свое выполнение.

Формат выходного файла

В начале взаимодействия с интерактором выведите номер игрока (1 или 2), за которого вы хотите играть. Учтите, что интерактор всегда играет оптимально и для решения задачи вам необходимо у него выиграть.

Далее выводите описания ходов по одному в строке (последовательность взаимодействия с интерактором описана в формате входных данных). После вывода каждой строки не забывайте делать flush.

Примеры

stdin	stdout
abacaba	1 1 4 2 2 5 2 1 6 1 7 1 3 -1
pp	2 1 2 1 1 -1

Примечание

Выходные данные к примерам содержат полный протокол взаимодействия решения с интерактором, т.е. первое число — номер игрока, далее по очереди идут ходы обоих игроков.

Разбор задачи L. Игра со строкой

В данной игре проигрышными являются те и только те состояния, когда каждая буква встречается в строке четное число раз. В них второй игрок имеет симметричную стратегию: он может каждым своим ходом стирать такой же набор букв, как и первый игрок предыдущим ходом. Из всех остальных состояний можно за один ход перейти в проигрышное, взяв те буквы, которые встречаются в строке нечетное число раз.

День седьмой (21.02.2014 г.) КонTEST Сергея Копелиовича

Об авторе...

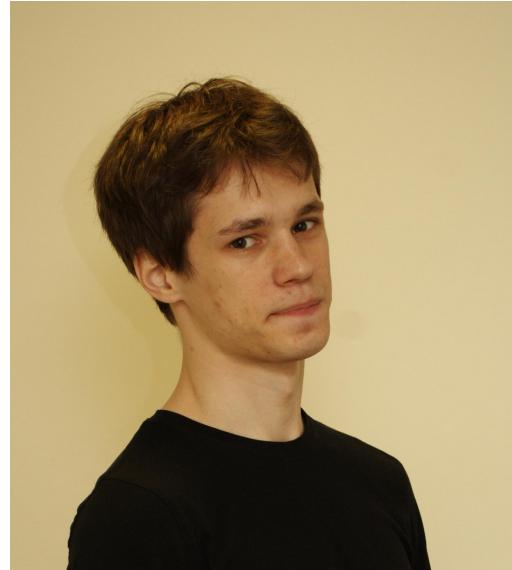
Копелиович Сергей Владимирович родился в Санкт-Петербурге в 1989 году. Закончил физ-мат лицей №30. В 4–6 классах занимался в кружке по математике. Олимпиадами по программированию начал активно заниматься в 9-м классе.

В июне 2012-го закончил МатМех СПбГУ. Сейчас студент 1-го курса магистратуры Санкт-Петербургского академического университета.

С марта 2012-го по январь 2013-го сотрудник ВКонтакте.

Постоянный участник сборов в Харькове (с 2011-го) и сборов в Петрозаводске (с 2004-го, как участник).

Член научного комитета Всероссийской школьной олимпиады по информатике и сборов по подготовке к международной олимпиаде школьников. В ACM соревнованиях выступал в команде Vifunduchki. Преподает с 2006-го года. Ведет школьные кружки, студенческие спецкурсы. Любит учить и учиться.



Основные достижения:

- Золотые медали на IOI в 2005 (Польша) и 2006 (Мексика) годах.
- Бронзовая и золотая медали в 2008 (Канада) и 2009 (Швеция) годах.
- Участвовал в Стокгольме в финале студенческого командного чемпионата мира по программированию ACM ICPC. Его команда заняла 3-е место.

Интересы: олимпиады, программирование, волейбол, фрисби, танцы.

Одна из любимых цитат: “Прежде чем в сто первый раз попытаться взлететь, забудь, что первые сто попыток тебя расстроили”.

На intuit.ru можно найти его старый курс (2008-й год) под названием “Базовые и “продвинутые” алгоритмы для школьников”.

Теоретический материал. Перебор и NP-полные задачи

Перебор и отсечения.

Текст ниже является лишь планом лекции. Полный текст будет доступен позже.

1. Начнем издалека: алгоритм A^*

- (a) Задача: Есть N точек на плоскости, некоторые точки соединены дорогами. Дорога — отрезок на плоскости.
- (b) Дейкстра: $\min d[start, v]$, $A^* : \min d[start, v] + r[v, end]$. Где r — расстояние на плоскости, а d — расстояние между вершинами в графе.
- (c) A^* , как и Дейкстра, останавливается, как только из кучи вынимается вершина end .
- (d) Дейкстра для графов с отрицательными весами ребер: алгоритм корректно работает, но дольше чем $E \log V$. Тем не менее алгоритм работает конечное время.
- (e) Переходим к перебору. Пусть наш граф — дерево, а мы хотим найти кратчайший путь до листа. Это легко сделать за $O(N)$, но мы хотим еще быстрее.
- (f) Придумаем оценку снизу $r[v, end]$ на $d[v, end]$ и запустим A^* , который переберет ровно все вершины, у которых $d[start, v] + r[v, end] \leq d[start, end]$.

2. Это глупо, но нужно об этом помнить

- (a) Отсечение по времени.
- (b) Предподсчет.

3. Нормальные отсечения на примере задачи о рюкзаке (у каждой вещи есть вес и ценность)

- (a) Сперва напишем рекурсивный перебор: каждую вещь или берем, или не берем. Получим решение за $O(2^n)$ в худшем случае.
- (b) Добавим запоминание. Получается ленивая динамика за $O(nW)$, где W — размер рюкзака.
- (c) Запоминание не всегда дает положительный эффект. Например, если в нашей задаче про рюкзак W большое, а веса вещей — случайные числа.

- (d) Придумаем жадность и с помощью нее получим оценки на ответ: отсортировали по убыванию частного $value_i/weight_i$, $X \leq d[v, end] \leq Y$.
- (e) Мы максимизируем суммарный вес, поэтому в отсечении по ответу используется: if ($d[start, v] + X \leq Best$) return. Переходы иногда выгодно делать в порядке Y .
- (f) Пусть мы угадали ответ. Тогда отсечение по ответу работает еще лучше.
- (g) Iterative deepening помогает усилить отсечение по ответу в случае дискретного ответа. В случае непрерывного может помочь бинпоиск по ответу.
- (h) После того, как у нас появились состояния, мы видим граф. На самом деле мы на этом графе ищем путь.
- (i) Почему мы ищем кратчайший (или длиннейший) путь на ациклическом графе рекурсивно?
 - i. Граф большой, алгоритм за $O(\text{размера графа})$ нам не подходит.
 - ii. Удобно пересчитывать состояния (рекурсия!).
 - iii. Лучше работает отсечение по ответу (если, конечно, нет Iterative deepening).
 - iv. Если нет запоминания, использует меньше памяти, $O(\text{глубины})$.
- (j) Можно искать путь “Дейкстрой” или “Форд-Беллманом с очередью”. Дейкстра лучше тем, что перебирает не больше вершин, чем перебор.
- (k) Как еще можно писать перебор? Можно несколько раз делать Random Walk. Пример для рюкзака: много раз начинаем с пустого набора и берем на каждом шаге случайный из 5 лучших по $value_i/weight_i$ кандидатов.

4. Еще пример для запоминания: бесплатно получаем динамику по изломанному профилю

- (a) Задача: сколько способов доску $w \times h$ разбить на доминошки?
- (b) Перебор: нашли первую не покрытую клетку и пытаемся ее покрыть двумя способами. Получили асимптотику $2^{wh/2}$. Важно: функция перебора возвращает сколько способов дойти до конца.
- (c) Отсечение: запоминание. Получили асимптотику $wh2^w$.

NP-трудные задачи (большая часть будет на разборе).

1. Клики, вершинные покрытия, независимые множества

- (a) Взаимные сведения (клика \Leftarrow независимое множество \Leftarrow вершинное покрытие).
- (b) Поиск максимальной подклиники и количества подкликов (а значит, и максимального независимого, и минимального покрытия) за $2^{n/2}n$. Насчитали для каждого множества “сколько в нем подкликов”.
- (c) Оптимизация $2^{n/2}n \rightarrow 2^{n/2}$. $s[p] = s[p \text{ xor } 2^x] + s[p \text{ and } c[x]]$. $m[p] = \max(m[p \text{ xor } 2^x], 1 + m[p \text{ and } c[x]])$.
- (d) Новая задача: поиск вершинного покрытия размера не более k .
- (e) Кернализация для вершинного покрытия: $(n, m) \rightarrow (k^2, k^2)$. Есть более крутая кернализация $(n, m) \rightarrow (3k, 3k)$.
- (f) Решение за $O(2^k k^2 + m)$ (непокрытое ребро нужно покрыть, есть 2 способа это сделать).
- (g) Решение за $O(1.61^k k^2 + m)$: $T(k) = T(k - 1) + T(k - \deg)$, $\deg \geq 2$, $\deg \geq 3$.
- (h) Более быстрые решения: $T(k) = \max(T(k - 1) + T(k - 4), T(k - 2) + T(k - 3))$. Итого 1.38^n .

2. Перебор всех максимальных по включению клик

- (a) Количество максимальных по включению клик не больше $3^{n/3} \approx 1.44^n$.
- (b) Доказательство. Ищем максимальное независимое множество. $T(n) \geq (d + 1)T(n - (d + 1))$ (выбрали вершину минимальной степени и расщепились по ней).
- (c) Алгоритм за $O(1.44^n)$ с полиномиальной памятью: поддерживаем множества вершин C (клика), X (вершины, которые должны с чем-то конфликтовать), P (вершины, которые еще можно добавить в клику)
- (d) Как доказать, что этот алгоритм работает за $O(\text{ответ})$? Непонятно.

3. Раскраска вершин графа в минимальное количество цветов

- (a) Алгоритм за $O(3^n)$. Динамика. $k[A]$, переход: $A = A_1 \coprod B$. Время работы = $\sum_k C_n^k 2^k = (1 + 2)^n$.
- (b) Алгоритм за $O(2.44^n)$. Та же динамика, но перебираем не все A_1 , а только максимальные по включению. Время работы = $\sum_k C_n^k 1.44^k = (1 + 1.44)^n$.

(c) Алгоритм за $O(2^n n)$ формулой включения-исключения.

i. Формула: $f[A] = \sum_{B \subset A} g[B] \Rightarrow g[A] = \sum_{B \subset A} (-1)^{|A \setminus B|} f[B]$.

ii. Будем считать, сколько способов покрыть V ровно k независимыми множествами. $f[A]$ — покрыть не обязательно все A . $g[A]$ — покрыть все A .

iii. $f[A] = m[A]^k$, $m[A] = m[A, n]$, где $m[A, 0] = is[A]$ и $[A, i] \rightarrow [A, i+1], [A \setminus 2^i, i+1]$.

iv. Получим алгоритм, который красит в минимальное количество цветов. Для этого сперва бинпоиском найдем k , а затем переберем x , что вершины 1 и x одного цвета, тогда их можно стянуть.

(d) Алгоритм за $O(2^n n^3)$ с помощью быстрого перемножения многочленов.

i. $P(x, y) = \sum_{is[A]=1} x^A y^{|A|}$

ii. $y = x^{2N}, N = 2^n$

iii. Считаем $P(x, y)^k$, смотрим на коэффициент при $x^{2^n-1} y^n$.

4. Раскраска вершин графа в 3 цвета

(a) Сперва научимся красить в 2 цвета dfs-ом за $O(E)$

(b) Теперь научимся красить в какой-то из двух допустимых цветов. Это $2-SAT$. Каждое ребро между вершинами, которые можно покрасить в один цвет — это условие.

(c) Жадное решение задачи, сформулированной в пункте (b): пусть вершина x_1 покрашена в цвет 1. Сделаем все возможные выводы, если нашли противоречие, то вершина x_1 должна быть покрашена в цвет 2, иначе оставляем цвет 1.

(d) Вероятностный алгоритм за $O(1.5^n)$. У каждой вершины можно запретить один из цветов. С вероятностью $(2/3)^n$ мы ни разу не ошибемся.

(e) Алгоритм за $O(1.44^n) \approx O(3^{n/3})$. В первый цвет покрасим максимальное по включению независимое множество.

(f) Алгоритм за $O(2^{n/2})$.

(g) Оптимальный алгоритм сейчас $O(1.3289^n)$.

5. Вероятностный алгоритм поиска пути длины k из a в b за $(2e)^k m$. Где $e \approx 2.718281828\dots$

(a) Покрасим вершины графа случайным образом в цвета от 1 до k .

- (b) Вероятность того, что все k вершин нашего пути покрашены в разные цвета равна $k!/k^k = 1/e^k$.
- (c) Найти разноцветный путь (а он обязательно простой) можно за $2^k m$.

6. Решение 3-SAT

- (a) Формулировка: дана конъюнкция из 3-дизъюнктов (клозов). Всего m клозов и n булевых переменных. Найти выполняющий набор.
- (b) $k\text{-SAT}(m, n) \rightarrow 3\text{-SAT}((k - 2)m, n + (k - 3)m)$
- (c) $O(2^n m)$ — для каждой переменной перебираем значение.
- (d) Оптимизируем до $O(2^n)$ — рекурсивно перебираем клозы.
- (e) $O(7^{n/3})$. Берем неудовлетворенный клоз и перебираем значения свободных переменных. 7 из 8 нам подойдут.
- (f) $O(3^{n/2}) \approx O(1.73^n)$. Берем неудовлетворенный клоз и перебираем значение свободных переменных. Заметим, что всегда есть клоз, в котором не более двух свободных переменных.
- (g) Вероятностный алгоритм за $O(3^{n/2}) \approx O(1.73^n)$. Рассмотрим ответ x . Пусть в нем не более $n/2$ единиц. Начинаем с $x_0 = \{0, 0, \dots, 0\}$. $n/2$ раз выбираем любой клоз, который не выполняется и у одной из переменных меняем значение на противоположное. С вероятностью $(1/3)^{n/2}$ мы из x_0 получим x .
- (h) Вероятностный алгоритм за $O((\frac{4}{3})^n) \approx O(1.33^n)$. Запускаем Random-Walk от случайного набора на глубину $3n$. Переход: выбираем случайный неудовлетворенный клоз и меняем в нем значение случайной переменной на противоположное. Получаем вероятность успеха $(\frac{3}{4})^n$.

7. Решение задачи про двудольный граф и k вершин

- (a) За n фаз решения такой задачи мы можем узнать, можно ли из произвольного графа G удалить k вершин так, чтобы он стал двудольным.
- (b) Переформулируем условие так: есть две доли и множество из k вершин.
- (c) Переберем, какие вершины из k добавленных (1) лежат в первой доле, (2) лежат во второй доле, (3) не войдут в двудольный подграф. Всего 3^k вариантов.
- (d) Теперь нужно часть (A) старого двудольного графа не перекрасить, часть (B) целиком перекрасить, а часть (C) выкинуть из графа так, чтобы между (A) и (B) не было ребер и $|C|$ было минимально.

- (e) Такое С можно найти с помощью минимального вершинного разреза.
- (f) Заметим, что разрез нас интересует величины не более k . Проверить, есть ли такой, а если есть, найти, можно за $O(mk)$.
- (g) Получили решение за $O(3^k km)$. Чтобы работало быстрее, лучше 3^k вариантов перебирать рекурсивно и на каждом шаге рекурсии проверять, что маленький разрез все еще есть.

Задачи и разборы

Задача А. Трисочетание

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	512 Мб

Дан трёхдольный полный взвешенный граф. Все три доли имеют одинаковый размер n . Трёхдольный означает, что нет ребер внутри доли. Полный означает, что между долями проведены все возможные ребра, каждое ребро ровно один раз. Нужно найти трисочетание минимального веса. Трисочетание — n троек индексов (a_i, b_i, c_i) такие, что все a_i различны, все b_i различны, все c_i различны. Вес трисочетания равен $\sum_{i=1..n} (w[0, a_i, b_i] + w[1, b_i, c_i] + w[2, c_i, a_i])$, где $w[i, a, b]$ — вес ребра из a -й вершины i -й доли в b -ю вершину $(i + 1) \bmod 3$ доли.

Формат входного файла

На первой строке число $n \geq 1$.

Следующие n строк содержат матрицу $w[0]$.

Следующие n строк содержат матрицу $w[1]$.

Следующие n строк содержат матрицу $w[2]$.

Гарантируется, что все веса — случайные числа от k до $2k$ для некоторого k . Все веса — целые числа от 0 до 10^5 .

Формат выходного файла

На первой строке выведите суммарный вес найденного трисочетания. На каждой из следующих n строк выведите $a_i b_i c_i$ (вершины нумеруются от 0 до $n - 1$). Тройки можно выводить в любом порядке. Если трисочетаний с минимальным суммарным весом несколько, подойдет любое.

Система оценки

Подзадача 1 (25 баллов) $n \leq 5$.

Подзадача 2 (25 баллов) $n \leq 9$.

Подзадача 3 (25 баллов) $n \leq 11$.

Подзадача 4 (25 баллов) $n \leq 13$.

Примеры

stdin	stdout
3	611
82 100 98	2 0 2
95 71 75	1 1 1
55 97 61	0 2 0
87 99 72	
92 59 72	
55 80 64	
54 57 90	
74 60 77	
92 50 87	

Разбор задачи А. Трисочетание

- 25 баллов. $(n!)^3 \cdot n$.
- 75 баллов. Динамика $[i_1, p_2, p_3]$, состояний не более 4^n , памяти 4^n т.к. $i = |p_1|$.
- 100 баллов. Локальные оптимизации венгерским алгоритмом поиска минимального по весу паросочетания в двудольном графе, начальное решение — случайное.

Задача В. Множество множеств

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	512 Мб

Даны несколько случайных подмножеств множества $A = \{1, 2, \dots, n\}$. Нужно выбрать минимальное количество подмножеств, которые в объединении дают A . Гарантируется, что объединение всех подмножеств равно A .

Формат входного файла

На первой строке число $n \geq 1$ и число множеств $m \geq 1$. Следующие m строк задают множества. Каждое множество задается числом элементов $k_i \geq 1$ и k_i различными числами от 1 до n . Гарантируется, что множества генерировались следующим образом: фиксируем $c \geq 5$, далее m раз сперва выбираем случайный размер s от 1 до c , далее равновероятно из всех C_n^s вариантов выбираем множество размера s .

Формат выходного файла

На первой строке выведите x — минимальное количество множеств. На следующей строке выведите номера выбранных множеств, x различных чисел от 1 до m . Номера можно выводить в произвольном порядке. Если оптимальных ответов несколько, выведите любой.

Система оценки

Подзадача 1 (25 баллов) $n \leq 20, m \leq 50$.

Подзадача 2 (25 баллов) $n \leq 30, m \leq 100$.

Подзадача 3 (25 баллов) $n \leq 40, m \leq 100$.

Подзадача 4 (25 баллов) $n \leq 50, m \leq 100$.

Примеры

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

Разбор задачи В. Множество множеств

1. 25 баллов. Динамика за $2^n m$.
2. 75 баллов. Будем расщепляться по множеству, покрывающему M максимальное количество непокрытых элементов. Добавим отсечение по ответу: каждое множество покроет не более M элементов.
3. 100 баллов. В решении на 75 баллов нужно. Добавить отсечение по времени. Также нужно модифицировать отсечение по ответу: взять $(\text{answer} - \text{current} - 1)$ максимальных M_i и сложить.
4. 100 баллов. В решении на 75 баллов нужно также добавить “перебор оптимальных К ребер” $K = 2, 3, 4\dots$ пока не TL .

Задача С. Длинная дорога

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 512 Мб

Дорога, дорога, осталось немного...

Дан случайный неориентированный граф G из n вершин и m ребер. Ваша задача — найти гамильтонов путь. Гарантируется, что гамильтонов путь в графе есть.

Формат входного файла

На первой строке число вершин $n \geq 2$ и число ребер $m \geq 1$.

Следующие m строк содержат пары чисел от 1 до n — ребра графа.

В графе нет ни петель, ни кратных ребер.

Поскольку почти полный граф — совсем не интересный тест, $m \leq 500$.

Формат выходного файла

На первой строке выведите n различных чисел от 1 до n — вершины гамильтонового пути в порядке прохода по ним. Начинать и заканчивать можно в любой вершине. Если гамильтоновых путей несколько, выведите любой.

Система оценки

Подзадача 1 (20 баллов) $n \leq 26$.

Подзадача 2 (20 баллов) $n \leq 35$.

Подзадача 3 (20 баллов) $n \leq 50$.

Подзадача 3 (20 баллов) $n \leq 70$.

Подзадача 4 (20 баллов) $n \leq 100$.

Примеры

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

Разбор задачи С. Длинная дорога

1. 20 и 40 баллов. Любой перебор. Можно добавить запоминание по вкусу.
2. 100 баллов. Начинаем со случайной вершины, много раз идем в случайную из соседних вершин минимальной степени.
3. Также возможно было отсекаться по критериям “остаток связан”, “дерево компонент вершинной двусвязности остатка не ветвится”. Но преимущества над предыдущим решением это не дает.

Задача Е. Корни

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 512 Мб

Дано целое число $n \geq 1$. Нужно найти такое g , что для любого a : $\gcd(a, n) = 1$, $1 \leq a < n$ $\exists x: g^x = a \pmod{n}$. Напомним, что $\gcd(a, b)$ — наибольший общий делитель чисел a и b .

Формат входного файла

Внимание, мультитест!

На каждой строке число n ($2 \leq n \leq 10^{12}$).

Сколько тестов, мы вам не скажем, но все в рамках приличия.

Формат выходного файла

Для каждого n на отдельной строке выведите g ($1 \leq g < n$) или -1 , если такого g не существует.

Система оценки

Подзадача 1 (15 баллов) $n \leq 10^3$.

Подзадача 2 (15 баллов) $n \leq 10^5$.

Подзадача 3 (35 баллов) $n \leq 10^9$.

Подзадача 4 (35 баллов) $n \leq 10^{12}$.

Примеры

stdin	stdout
5	2
10	3
9	2
15	-1

Разбор задачи Е. Корни

- 15 баллов. Проверили за линейное время все $g : (g, n) = 1$.
- 30 баллов. g не больше 40 (порядка $\log n$). В предположении гипотезы Римана о нулях дзета-функции доказано, что $g = O(\log^6 p)$ для простых p . На практике видим, что $\exists g = O(\log n)$.
- 65 баллов. $(g, n) = 1 \Rightarrow g^{\varphi(n)} = 1(n)$ (теорема Эйлера), а также если $g^x = 1(n)$, то $x \mid \varphi(n)$. Поэтому каждое число можно проверять за $O(\log n) : g^{\varphi(n)/p} \neq 1$ для простых делителей p числа $\varphi(n)$.
- 100 баллов. Чтобы найти $\varphi(n)$ и потом перебирать его простые делители, нужно разложить на множители n , а затем $\varphi(n)$. Это можно делать за корень из n , а можно за количество простых до корня из n (в $\log n$ раз меньше).
- Для ускорения и простоты реализации можно использовать знание, что g существует для n вида p^k и $2p^k$, где p — простое нечетное, а также для $n = 2$ и $n = 4$.

Задача F. Умножение многочленов

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	512 Мб

Даны многочлены $P(x)$ и $Q(x)$, найдите $P(x)Q(x)$. $Q(x) = x^k - 1$.

У $P(x)$ все коэффициенты — случайные целые числа от 0 до $m - 1$.
Все вычисления происходят по модулю m , m — простое от 2 до $10^9 + 7$.

Формат входного файла

Первая строка содержит целые числа n, k, m .

Следующая строка содержит n чисел p_0, p_1, \dots, p_{n-1} , $P(x) = \sum p_i x^i$.

Формат выходного файла

Выведите $n + k$ целых чисел от 0 до $m - 1$: $b_0, b_1, \dots, b_{n+k-1}$,
такие, что $P(x)Q(x) = \sum b_i x^i \pmod{m}$.

Система оценки

Подзадача 1 (50 баллов) $1 \leq n, k \leq 10^3$.

Подзадача 2 (50 баллов) $1 \leq n, k \leq 10^5$.

Примеры

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

Разбор задачи F. Умножение многочленов

- 100 баллов. Возьмем и умножим циклом `for`. Заметим, что в общем случае два многочлена P и Q можно умножить за $f(P) \cdot f(Q)$, где f — количество ненулевых коэффициентов.
- Для скорости реализации также заметим, что нужно только складывать и вычитать.

Задача G. Деление многочленов

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	512 Мб

Даны многочлены $P(x)$ и $Q(x)$, найдите такие $A(x)$ и $R(x)$, что $P(x) = A(x)Q(x) + R(x)$ и $\deg R < \deg Q$. $Q(x) = x^k - 1$. У $P(x)$ все коэффициенты — случайные целые числа от 0 до $m - 1$. Все вычисления происходят по модулю m , m — простое от 2 до $10^9 + 7$.

Формат входного файла

Первая строка содержит целые числа n, k, m .

Следующая строка содержит n чисел p_0, p_1, \dots, p_{n-1} , $P(x) = \sum p_i x^i$.

Формат выходного файла

На первой строке выведите t целых чисел от 0 до $m - 1$: a_0, a_1, \dots, a_{t-1} . Где $t = \max(n - k, 1)$. На второй строке k целых чисел от 0 до $m - 1$: r_0, r_1, \dots, r_{k-1} . Все эти числа должны обладать свойством, что $P(x) = Q(x)(\sum a_i x^i) + \sum r_i x^i \bmod m$.

Система оценки

Подзадача 1 (50 баллов) $1 \leq n, k \leq 10^3$.

Подзадача 2 (50 баллов) $1 \leq n, k \leq 10^5$.

Примеры

stdin	stdout
3 1 7	6 1
1 5 1	0
3 1 7	5 1
2 4 1	0
3 1 7	5 1
0 4 1	5

Разбор задачи G. Деление многочленов

- 100 баллов. Последовательно вычитаем $a_n x^{n-k} (x^k - 1)$ из $P(x)$, где старший моном $P(x)$ равен $a_n x^n$. Каждое такое вычитание работает за $O(1)$ и уменьшает длину P на 1.

Задача H. Ребра добавляются, граф растет

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 512 Мб

В неориентированный граф последовательно добавляются новые ребра. Изначально граф пустой. После каждого добавления нужно говорить, является ли текущий граф двудольным.

Формат входного файла

На первой строке n — количество вершин, m — количество операций “добавить ребро”. Следующие m строк содержат пары чисел от 1 до n — описание добавляемых ребер.

Формат выходного файла

Выведите в строчку m нулей и единиц. i -й символ должен быть равен единице, если граф, состоящий из первых i ребер, является двудольным.

Система оценки

Подзадача 1 (25 баллов) $1 \leq n, m \leq 1\,000$.

Подзадача 2 (50 баллов) $1 \leq n, m \leq 50\,000$.

Подзадача 3 (25 баллов) $1 \leq n, m \leq 300\,000$.

Примеры

stdin	stdout
3 3 1 2 2 3 3 1	110

Разбор задачи Н. Ребра добавляются, граф растет

- 75 баллов. Бинарный поиск по ответу и раскраска в два цвета.
- 100 баллов. Система непересекающихся множеств (CHM, DSU).

Задача I. Сумма всего подряд

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	128 Мб

Дан случайный неориентированный граф. Нужно для каждого множества вершин A посчитать $f(A)$, количество независимых подмножеств вершин B : $B \subseteq A$. Множество вершин B называется независимым, если в графе нет ребра, оба конца которого лежат в множестве B .

Формат входного файла

На первой строке число вершин $n \geq 1$ и число ребер $m \geq 1$.

Следующие m строк содержат пары чисел от 1 до n — ребра графа.
В графе нет ни петель, ни кратных ребер.

Формат выходного файла

Каждому множеству A можно сопоставить целое число $b(A)$, двоичная запись которого соответствует наличию элементов в множестве A .

Пример: $n = 5, A = \{1, 2, 5\}, b(A) = 2^0 + 2^1 + 2^4 = 19$.

Выведите $\sum_A f(A)2^{b(A)} \pmod{10^9 + 7}$

Система оценки

Подзадача 1 (25 баллов) $1 \leq n \leq 10$.

Подзадача 2 (25 баллов) $1 \leq n \leq 16$.

Подзадача 3 (30 баллов) $1 \leq n \leq 20$.

Подзадача 4 (20 баллов) $1 \leq n \leq 23$.

Примеры

stdin	stdout
3 1	1221
1 2	

Пояснение

Независимыми являются множества вершин $\{\}, \{1\}, \{2\}, \{3\}, \{1, 3\}, \{2, 3\}$.

$$A = \{\} \quad f(A) = 1 \quad b(A) = 0$$

$$A = \{1\} \quad f(A) = 2 \quad b(A) = 2^0 = 1$$

$$A = \{2\} \quad f(A) = 2 \quad b(A) = 2^1 = 2$$

$$A = \{1, 2\} \quad f(A) = 3 \quad b(A) = 2^0 + 2^1 = 3$$

$$A = \{3\} \quad f(A) = 2 \quad b(A) = 2^2 = 4$$

$$A = \{1, 3\} \quad f(A) = 4 \quad b(A) = 2^0 + 2^2 = 5$$

$$A = \{2, 3\} \quad f(A) = 4 \quad b(A) = 2^1 + 2^2 = 6$$

$$A = \{1, 2, 3\} \quad f(A) = 6 \quad b(A) = 2^0 + 2^1 + 2^2 = 7$$

$$1 \cdot 2^0 + 2 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + 2 \cdot 2^4 + 4 \cdot 2^5 + 4 \cdot 2^6 + 6 \cdot 2^7 = 1221$$

Разбор задачи I. Сумма всего подряд

- 25 баллов. Переберем A , переберем B (итого 4^n), если B — подмножество A и B независимо, то $f[A]++$.
- 50 баллов. Переберем A , переберем все его подмножества B (итого 3^n), если B — подмножество A и B независимо, то $f[A]++$.

3. 100 баллов. Пусть i лежит в A , $f[A] = f[A \oplus 2^i] + f[A \& c[i]]$, где $c[i]$ — множество вершин, с которым A не смежно.

Задача J. Все клики

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	512 Мб

Макс, опытнейший игрок в доту, все кликал и кликал.

Дан случайный неориентированный граф G из n вершин и m ребер. Подкликой называется такое подмножество вершин A : $\forall a, b \in A, a \neq b \exists$ ребро (a, b) . Подклика A называется максимальной по включению, если \nexists подклики $B \supset A$: $|B| > |A|$. $K(G)$ — количество максимальных по включению подкликов в графе G . Ваша задача — посчитать $\min(K(G), 10^6)$.

Формат входного файла

На первой строке число вершин $n \geq 1$ и число ребер $m \geq 1$.

Следующие m строк содержат пары чисел от 1 до n — ребра графа.

В графе нет ни петель, ни кратных ребер.

Формат выходного файла

Единственное число — $\min(K(G), 10^6)$.

Система оценки

Подзадача 1 (15 баллов)	$n = 20$.
Подзадача 2 (15 баллов)	$n = 25$.
Подзадача 3 (20 баллов)	$n = 30$.
Подзадача 4 (15 баллов)	$n = 40$.
Подзадача 5 (20 баллов)	$n = 50$.
Подзадача 6 (15 баллов)	$n = 60$.

Примеры

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

Разбор задачи J. Все клики

1. 30 баллов. Напишем рекурсивный перебор всех подмножеств. Множества удобно хранить (пересекать, объединять) в `int64`. Чтобы проверить, что клика максимальна по включению, можно за линию попробовать в нее добавить каждую вершину.
2. 85 баллов. В рекурсивном переборе, если мы не берем вершину, она должна конфликтовать с какой-то вершиной, которую мы не берем. Храним множество X — те вершины, которые должны конфликтовать, но еще не конфликтуют. Если какая-то вершина из X смежна со всеми вершинами, которые мы можем добавить, то `return`.
3. 100 баллов. Если какая-то вершина смежна со всеми вершинами, которые мы еще можем добавить, ее нужно жадно взять. Расщепляться будем по вершине минимальной степени.

Задача K. Макс клика

Вход: **stdin**
 Выход: **stdout**
 Ограничение по времени: 0.5 с
 Ограничение по памяти: 512 Мб

Макс, опытнейший игрок в доту, постоянно кликал.

Дан случайный неориентированный граф G из n вершин и m ребер.
 Подкликой называется такое подмножество вершин A : $\forall a, b \in A, a \neq b \quad \exists$

ребро (a, b) .

Ваша задача — найти подклику A : $|A|$ максимально.

Формат входного файла

На первой строке число вершин $n \geq 1$ и число ребер $m \geq 1$.

Следующие m строк содержат пары чисел от 1 до n — ребра графа.

В графе нет ни петель, ни кратных ребер.

Формат выходного файла

На первой строке выведите k — количество вершин в максимальной подклике. На следующей строке k целых чисел от 1 до n — номера вершин в подклике. Вершины можно выводить в любом порядке. Если максимальных подкликов несколько, выведите любую.

Система оценки

Подзадача 1 (10 баллов) $n \leq 20$.

Подзадача 2 (10 баллов) $n \leq 25$.

Подзадача 3 (20 баллов) $n \leq 40$.

Подзадача 4 (20 баллов) $n \leq 60$.

Подзадача 5 (20 баллов) $n \leq 70$.

Подзадача 6 (20 баллов) $n \leq 80$.

Примеры

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

Разбор задачи К. Макс клика

1. 20 баллов. Перебор всех подмножеств.

2. 40 баллов. Разбили на две половины и решили за $2^{n/2}$.

3. 60 баллов. Перебор с отсечением по ответу “сколько уже набрали + сколько еще можем добавить \leq максимальный ответ”.
4. 80 баллов. Если есть вершина максимальной степени или максимальной степени $- 1$, ее выгодно взять в максимальную клику.
5. 100 баллов. Добавим отсечение по времени =).

Задача L. Учимся красить

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	512 Мб

Дан случайный неориентированный граф G из n вершин и m ребер. Ваша задача — покрасить его вершины в минимальное количество цветов таким образом, чтобы смежные вершины были покрашены в разные цвета.

Формат входного файла

На первой строке число вершин $n \geq 1$ и число ребер $m \geq 1$.

Следующие m строк содержат пары чисел от 1 до n — ребра графа.

В графе нет ни петель, ни кратных ребер.

Формат выходного файла

На первой строке выведите k — минимальное количество цветов. На следующей строке n целых чисел от 1 до k — цвета вершин. Если оптимальных раскрасок несколько, выведите любую.

Система оценки

- | | |
|--------------------------------|---------------|
| Подзадача 1 (10 баллов) | $n \leq 15$. |
| Подзадача 2 (15 баллов) | $n \leq 20$. |
| Подзадача 3 (15 баллов) | $n \leq 30$. |
| Подзадача 4 (15 баллов) | $n \leq 40$. |
| Подзадача 5 (15 баллов) | $n \leq 50$. |
| Подзадача 6 (15 баллов) | $n \leq 60$. |
| Подзадача 7 (15 баллов) | $n \leq 70$. |

Примеры

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

Разбор задачи L. Учимся красить

- 10 баллов. Решение за 3^n . Динамика из лекции.
- 25 баллов. Будем последовательно красить вершины в цвета от 0 до $k - 1$ или в новый цвет, где k — количество уже использованных цветов. В новый цвет — в последнюю очередь.
- 70 баллов. Красим также, как и выше, в каждый момент времени красим ту вершину, у которой больше разноцветных соседей.
- 85 баллов. Выгодно на каждом шаге среди вершин с одинаковым количеством разноцветных соседей жадно выбирать случайную и расщепляться по ней. Запустим этот перебор на M итераций несколько раз. Запускаем, пока не TL. M подбирается.
- 100 баллов. Решение отсутствует.

Задача M. Проще всего красить в три цвета

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 0.5 с
Ограничение по памяти: 512 Мб

Дан случайный неориентированный граф G из n вершин и m ребер. Ваша задача — покрасить его вершины в три цвета таким образом, чтобы смежные вершины были покрашены в разные цвета. Гарантируется, что покрасить граф в три цвета возможно.

Формат входного файла

На первой строке число вершин $n \geq 1$ и число ребер $m \geq 1$.

Следующие m строк содержат пары чисел от 1 до n — ребра графа.

В графе нет ни петель, ни кратных ребер.

Формат выходного файла

Вывести n целых чисел от 1 до 3 — цвета вершин. Если требуемых раскрасок несколько, выведите любую.

Система оценки

Подзадача 1 (10 баллов) $n \leq 20$.

Подзадача 2 (15 баллов) $n \leq 30$.

Подзадача 3 (10 баллов) $n \leq 45$.

Подзадача 4 (15 баллов) $n \leq 64$.

Подзадача 5 (15 баллов) $n \leq 90$.

Подзадача 6 (10 баллов) $n \leq 120$.

Подзадача 7 (10 баллов) $n \leq 170$.

Подзадача 8 (15 баллов) $n \leq 250$.

Примеры

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

Разбор задачи М. Проще всего красить в 3 цвета

1. 25 баллов. Переберем вершины первого цвета (независимое множество), покрасим в два цвета.
2. 35 баллов. Переберем вершины первого цвета, в каждый момент рекурсии проверяем, что те вершины, которые остались образуют двудольный граф.
3. 85 баллов. Красим вершины в 3 цвета, в каждый момент времени красим ту вершину, у которой больше разноцветных соседей.

4. 100 баллов. Нужно перебирать максимальное по включению множество. Выгодно среди вершин с одинаковым количеством разноцветных соседей выбирать случайную. Запустим этот перебор на 30 000 итераций несколько раз. Запускать будем, пока не TL.

Задача N. SAT USAT

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	0.5 с
Ограничение по памяти:	512 Мб

Широко известна задача **3-SAT**. Решите её. Гарантируется, что решение существует. Формулировка **3-SAT**: даны m клозов, состоящих из трех условий вида $x_i = 0$ и $x_i = 1$, нужно подобрать значения n булевых переменных таким образом, чтобы в каждом клозе хотя бы одно из трёх условий было верно.

Формат входного файла

На первой строке число переменных n и клозов m ($1 \leq m \leq \min(n^2, 1000)$).

Каждая из следующих m строк содержит числа i, e_1, j, e_2, k, e_3 и задаёт условия $x_i = e_1 \vee x_j = e_2 \vee x_k = e_3$. $0 \leq e_1, e_2, e_3 \leq 1; 1 \leq i, j, k \leq n$. **Все клозы случайны, тем не менее гарантируется, что решение существует.**

Формат выходного файла

Выведите строку из n нулей и единиц — значения переменных. Если у данной задачи **3-SAT** есть несколько решений, выведите любое.

Система оценки

- Подзадача 1 (10 баллов)** $n \leq 20$.
- Подзадача 2 (15 баллов)** $n \leq 30$.
- Подзадача 3 (15 баллов)** $n \leq 40$.
- Подзадача 4 (15 баллов)** $n \leq 50$.
- Подзадача 5 (15 баллов)** $n \leq 70$.
- Подзадача 6 (15 баллов)** $n \leq 100$.
- Подзадача 7 (15 баллов)** $n \leq 160$.

Примеры

stdin	stdout
<pre> 2 3 1 0 1 0 1 0 2 0 2 1 1 1 1 1 2 1 1 1 </pre>	01

Пояснение

$$\begin{aligned}
 & \left(\underline{x_1 = 0} \vee \underline{x_1 = 0} \vee \underline{x_1 = 0} \right) \wedge \\
 & \left(\underline{x_2 = 0} \vee \underline{x_2 = 1} \vee x_1 = 1 \right) \wedge \\
 & \left(x_1 = 1 \vee \underline{x_2 = 1} \vee x_1 = 1 \right)
 \end{aligned}$$

В каждом клозе подчеркнуты истинные условия.

Разбор задачи N. SAT USAT

- 10 баллов. Перебор всех решений
- 70 баллов. Много раз подряд выбираем случайное решение и 3n раз перебираем все неудовлетворенные клозы и меняем значение случайной из трех переменных.
- 85 баллов. Подогнать константу 3n, еще можно пробовать начинать не со случайного решения, а лучшего из 100 случайных (минимальное число клозов не выполнены). Еще можно делать не $const \times n$ раз, а по n раз, пока количество невыполненных клозов уменьшается.
- 100 баллов. с у меня нет решения на 100 баллов =).

Задача O. Почти двудольный зверь

Вход:	stdin
Выход:	stdout
Ограничение по времени:	0.5 с
Ограничение по памяти:	512 Мб

Дан неориентированный, неслучайный граф. Вершины графа разбиты на три множества A , B , X . $|X| = k \leq 10$. Между вершинами из A нет ребер, между вершинами из B также нет ребер. Ваша задача выделить минимальное по размеру множество вершин Y , что после выкидывания этого множества граф становится двудольным. Заметьте, что $|Y| \leq k$, так как при выкидывании X граф точно станет двудольным.

Формат входного файла

На первой строке число вершин $n \geq 1$, число ребер $m \geq 1$ и $k \geq 1$ — размер множества X , $klen$. Следующие m строк содержат пары чисел от 1 до n — ребра графа. В графе нет ни петель, ни кратных ребер. Вершины из A и B нумеруются числами от 1 до $n - k$. Вершины множества X нумеруются от $n - k + 1$ до n . Номера вершин A и B перемешаны. Специально, чтобы Вам было... удобно.

Формат выходного файла

В первой строке выведите l — размер множества Y , на второй строке l различных целых чисел от 1 до n (элементы множества Y). Если оптимальных множеств несколько, выведите любое.

Система оценки

Подзадача 1 (20 баллов) $n \leq 20, k \leq 10, m \leq 90$.

Подзадача 2 (40 баллов) $n \leq 100, k \leq 10, m \leq 200$.

Подзадача 3 (40 баллов) $n \leq 1000, k \leq 10, m \leq 2000$.

Примеры

stdin	stdout
7 10 2 1 6 1 3 1 7 2 3 2 7 3 4 3 6 4 5 4 7 5 6	1 1
9 15 2 1 2 1 3 1 5 1 7 2 4 3 4 3 6 4 8 4 9 5 9 5 8 6 8 6 9 7 8 7 9	1 1

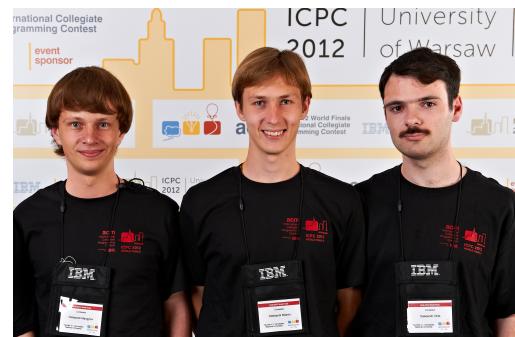
Разбор задачи О. Почти двудольный зверь

1. 20 баллов. Перебор всех подмножеств.
2. 100 баллов. $O(3^k km)$, смотри лекцию.

День восьмой (22.02.2014 г.) Конкурс Akai

Об авторе...

Миланин Александр, родился 28 августа 1988 года в Донецке. В 2010 году окончил Таврический национальный университет им. В. И. Вернадского по специальности “прикладная математика”. С 2010 г. по 2012 г. работал в компании Extensoft в проекте Artisteer. С 2010 г. работает ассистентом кафедры Информатики в Таврическом национальном университете.



Основные достижения:

- 2 место на полуфинале ACM ICPC в 2011 в Бухаресте.
- Финалист ACM ICPC World Finals 2012.
- Финалист Russian Code Cup в 2011, 2012 и 2013.
- 3 место в Russian AI Cup в 2012.
- 4 место в Russian AI Cup в 2013.
- Финалист TopCoder Open в номинации Marathon в 2013.

Интересы: футбол, теннис.

Теоретический материал. Поиск минимального элемента в стеке, в очереди и в скользящем окне в массиве

Рассмотрим стек. В стеке искать минимум просто. Обычное динамическое программирование. Вместе с основным стеком будем дополнительно поддерживать ещё один стек, в котором будет храниться минимум из всех элементов, которые сейчас в стеке. Добавляя новый элемент нам надо получить новый минимум. Он будет равен минимуму из текущего минимума и добавляемого элемента. Удаляя элемент, просто удалим самый верхний минимум.

- Стек: 5, 3, 7, 6, 2, 5

- Минимумы: 5, 3, 3, 3, 2, 2

Перейдём к очереди. Очередь можно симулировать с помощью двух стеков. Заведём два стека. Изначально они будут пустыми. Добавляя элемент, будем добавлять его в первый стек. Удаляя элемент будем удалять его из второго. Но второй иногда бывает пустым. Если на момент удаления элементов второй стек пустой, то переложим все элементы из первого стека во второй. При этом порядок этих элементов поменяется на обратный, то есть, удаляя элементы с верхушки второго стека мы по сути удаляем элементы снизу первого стека, что эквивалентно удалению элементов из конца очереди.

$$\begin{array}{ll} \text{Стек1: } & \{1, 2, 3, 4, 5\} \xrightarrow{\text{pop()}} \{\} \\ \text{Стек2: } & \{\} \xrightarrow{\text{push(6)}} \{6\} \\ & \{5, 4, 3, 2\} \end{array}$$

Заметим, что каждый элемент будет вставлен в первый стек ровно один раз, ровно один раз перенесён в другой стек и ровно один раз вынут из второго. То есть с каждым элементом будет осуществлено только $O(1)$ операций. Значит, амортизированная сложность будет $O(1)$. Понятие “амортизированная сложность” означает среднее число операций. Хотя в некоторых ситуациях одна отдельно взятая операция (в данном случае операция `pop()`, когда требуется переложить элементы из стека в стек) может выполняться асимптотически дольше. Для поиска минимума в очереди будем искать минимум в каждом из двух стеков. Поиск минимума в скользящем окне массива осуществляется путём эмуляции окна очередью.

Поиск элемента в матрице все строки и столбцы которой отсортированы.

Дана матрица, все строки и столбцы отсортированы. Например:

$$\begin{pmatrix} 1 & 3 & 8 \\ 2 & 5 & 9 \\ 4 & 7 & 9 \end{pmatrix}$$

Очевидно, что можно применить двоичный поиск к каждой строке или каждому столбцу. Сложность будет $O(n \log(m))$. Но можно найти элемент за время $O(n + m)$. Начнём поиск с нижнего левого элемента массива. С элемента $a[n - 1][0]$. Пусть $I = n - 1$, $j = 0$. Будем искать элемент x . Алгоритм поиска:

```
1 while (a[i][j] != x && i >= 0 && j < m) {
2             if (a[i][j] == x) return (i, j);
3             if (a[i][j] < x) j++;
4             else i--;
5 }
6 return -1;
```

Доказательство корректности оставим читателю.

Поиск отсутствующих элементов в перестановке.

Дана перестановка из n элементов. Некоторые из них заменены на -1 . Определить, какие элементы заменены.

Если можно использовать дополнительную память — всё просто. Заведём булевый массив $used[n]$ и отметим в нём все элементы из нашего массива. Потом выведем все неотмеченные элементы. Сложность $O(n)$. Но это неинтересно. Введём дополнительное ограничение. Можно использовать только $O(1)$ дополнительной памяти. Очевидное решение в этом случае — отсортировать перестановку и рассмотреть все соседние элементы. Если разность между ними не равна 1, значит некоторые пропущены. Сложность $O(n \log(n))$. Довольно медленно.

Пусть отсутствует ровно один элемент. Заметим, что упорядоченные элементы перестановки представляют собой арифметическую прогрессию с первым элементом 1 и разностью тоже 1. То есть, мы можем за константу посчитать их сумму. Пройдёмся по перестановке и вычтем из суммы все присутствующие элементы. Очевидно, что получим отсутствующий. Сложность $O(n)$.

Пусть отсутствует произвольное число элементов. Будем идти по массиву с перестановкой. Если текущий элемент массива $a[i] \neq -1$ и не стоит на своём месте ($a[i] \neq i$), то давайте поставим его на своё место: $a[a[i]] = a[i]$. При этом, если $a[a[i]] \neq -1$, нам тоже надо будет поставить его на своё место. Если же $a[a[i]] = -1$, то запишем его вместо $a[i]$. Сделаем это в цикле. Заметим, что каждый элемент, будучи поставлен на своё место, больше никуда не сдвинется, и при этом каждый элемент ставится на своё место за одну операцию. Значит, мы за $O(n)$ поставим все присутствующие элементы на своё место. Все позиции, на которых окажутся -1 соответствуют отсутствующим элементам.

Поиск мажорирующего элемента.

Мажорирующим элементом на массиве из n элементов называется элемент, который встречается строго более $\frac{n}{2}$ раз. Очевидно, что этот элемент является медианой массива. Медиану можно, очевидно, найти сортировкой, но это будет $O(n \log(n))$. Можно найти специальным линейным алгоритмом

поиска медианы, но он довольно сложный. Можно проще.

Заметим, что если элемент x — мажорирующий на массиве A , то для любого разбиения массива на префикс P и суффикс S , элемент x будет мажорирующим либо на префиксе, либо на суффиксе, либо на обоих. Очевидно доказывается от противного. Воспользуемся этим свойством. Будем идти по массиву слева направо. Для текущего префикса i будем хранить мажорирующий элемент на этом префикссе, если он есть, и число раз, которое он встретился. Рассмотрим префикс i . Если у нас на нём нет мажорирующего элемента, то просто отбросим этот префикс, так как по утверждению выше мажорирующий элемент на всем массиве — также мажорирующий на суффиксе $i + 1$ и можно найти его там. Если же мажорирующий элемент есть, то, если текущий элемент $a[i]$ равен мажорирующему, то обновим счётчики и перейдём к следующему шагу. Иначе просто увеличим длину префикса. Если после изменения длины префикса мажорирующий элемент перестал быть мажорирующим, скажем, что на текущем префиксе мажорирующего элемента нет, и будем искать его на суффиксе.

Задачи и разборы

Задача A. Anti-Lines (высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	5 с
Ограничение по памяти:	256 Мб

В этой задаче речь пойдет об игре “Lines”. В классической версии есть квадратное поле размером 9×9 клеток. Каждая клетка может быть либо пустой, либо содержать шарик одного из 7 цветов. Каждый ход компьютер выставляет три шарика случайных цветов в случайные пустые клетки. После чего Игрок может переместить один любой шарик на поле в какую-то допустимую пустую клетку. Если 5 или более шариков одного цвета расположить в виде вертикальной, горизонтальной или диагональной линии, то эти шарики уничтожаются, а игрок получает очки и дополнительный ход.

В этой задаче все происходит немного иначе: мы взломали компьютер и на каждом ходу можем выбирать пустые клетки, в которые выставляются новые шарики и цвета этих шариков. Каждый ход выставляется ровно по два новых шарика. Игрок ничего не передвигает (поскольку мы уже руководим действиями компьютера). Цель — совершать такие ходы, чтобы в момент первого уничтожения шариков количество одновременно уничтожившихся шариков было максимальным. То есть, можно совершить несколько ходов, собрать

какую-нибудь удачную конструкцию, а потом одним ходом ее уничтожить. Делать дополнительные уничтожения, очищая при этом карту, нельзя.

Два шарика выставляются на поле одновременно. После этого любой шарик, который принадлежит вертикальной, горизонтальной или диагональной линии хотя бы из 5 шариков одного цвета, мгновенно уничтожается. Если на поле меньше двух пустых клеток, то ход совершить нельзя.

Формат входного файла

Ровно 9 строк по 9 символов каждая — текущая позиция в игре. Каждый символ — это либо цифра от 1 до 7, означающая цвет шарика в текущей клетке, либо символ ‘.’, если клетка пустая. Гарантируется, что на поле нет ни одной линии из 5 или более шариков одного цвета.

Формат выходного файла

В первой строке два числа D и M — максимальное количество шариков, которые можно уничтожить одним ходом, и количество ходов, которые приводят к этому уничтожению. Далее $2M$ строк по две на каждый ход. Каждая строка должна содержать три числа r_i , c_i и $color_i$ ($1 \leq r_i, c_i \leq 9$, $1 \leq color_i \leq 7$) — номер строки и столбца пустой клетки, на которую надо выставить шарик цвета $color_i$. Оба шарика одного хода выставляются одновременно. Ровно D шариков должны уничтожиться в процессе первого уничтожения (теоретически можно уничтожать несколько раз и делать лишние ходы, но это делать не рекомендуется).

Если ни одного шарика уничтожить нельзя, то D должно быть равно 0.

Примеры

stdin	stdout
723134552	25 9
2421.2456	6 7 2
353...442	3 4 2
14...2..4	5 8 2
23.2....4	5 7 2
4....1..1	5 3 2
1222..112	6 3 2
3621..124	3 6 2
631211777	8 5 2
	7 5 2
	6 5 2
	5 5 2
	2 5 2
	3 5 2
	4 8 2
	4 7 2
	4 4 2
	5 6 2
	4 5 2

Разбор задачи A. Anti-Lines

Задача решается перебором. Зафиксируем последний ход. То есть две клетки и цвета попадающих в них шариков. Эти две клетки являются центрами уничтожения шариков. Поэтому нужно перебрать, какие из клеток в восьми направлениях будут содержать шарики того же цвета, что и центр. Важно, чтобы при формировании конструкции, которая уничтожится на последнем ходу, не возникло случайно линии из пяти шариков, которые бы уничтожились раньше, чем наступит последний ход. Именно поэтому конструкцию нельзя строить жадно.

Однако, в процессе перебора используется жадная оценка сверху на то, какое количество шариков можно уничтожить. Если оказывается, что текущее количество шариков в конструкции плюс жадная оценка оказались меньше, чем текущий оптимальный ответ, эта ветка перебора отсекается.

Одно из важных отсечений заключается в том, что если зафиксировать некоторый ответ (метод iterative deepening), то можно сделать заключение, что один из двух центров уничтожения инициирует хотя бы половину всех уничтоженных шаров. Таким образом отсекаются тяжелые ветки перебора, когда

первым центром было уничтожено мало шаров, но жадность все еще утверждает, что вторым центром можно уничтожить много. А также, если при переборе первого центра уже поставлено на карту много шаров, то перебор шаров второго центра существенно ускоряется за счет большого количества уже зафиксированной информации на карте.

Задача В. Big Fellow (высшая лига)

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

На день рождения Вам подарили игру *пти-нашки*. Она представляет собой таблицу, состоящую из n строк и m столбцов. В каждой её ячейке лежит карточка с натуральным числом. Когда все карточки лежат на своём месте, таблица выглядит так:

$$\begin{array}{cccccc} 1 & 2 & 3 & \dots & m \\ m+1 & m+2 & m+3 & \dots & 2m \\ 2m+1 & 2m+2 & 2m+3 & \dots & 3m \\ \dots & \dots & \dots & \dots & \dots \\ (n-1)m+1 & (n-1)m+2 & (n-1)m+3 & \dots & nm \end{array}$$

Как-то Вы пришли домой и увидели, что с таблицей кто-то поигрался. Она стала выглядеть так:

$$\begin{array}{cccccc} 1 & n+1 & 2n+1 & \dots & (m-1)n+1 \\ 2 & n+2 & 2n+2 & \dots & (m-1)n+2 \\ 3 & n+3 & 2n+3 & \dots & (m-1)n+3 \\ \dots & \dots & \dots & \dots & \dots \\ n & 2n & 3n & \dots & mn \end{array}$$

Вам ужасно не понравился новый вид таблицы, и Вы решили уложить все карточки на место. Для этого Вы решили выполнить последовательность операций. Каждая операция состоит из следующих действий:

- 1) Поднять какую-либо карточку в воздух
- 2) Найти её правильное место в таблице

- 3) Если это место свободно, положить туда карточку, находящуюся в воздухе
- 4) Иначе, поменять местами карточку, лежащую на этом месте, с карточкой, находящейся в воздухе, и вернуться к шагу 2.

При этом Вы решили поставить себе условие, что в процессе укладывания каждая из m карточек должна побывать в воздухе хотя бы один раз. Это означает, что даже если карточка уже лежит на своём месте, Вам придётся потратить операцию на её поднятие и укладывание обратно.

Какое минимальное количество таких операций Вам необходимо выполнить для приведения таблицы к исходному виду?

Формат выходного файла

Вывести единственное число – минимальное количество необходимых операций.

Примеры

stdin	stdout
4 4	10
2 3	3

Разбор задачи B. Big Fellow

Давайте нумеровать элементы и индексы с нуля. Заметим, что нам надо найти количество циклов в перестановке $0, m, 2m, \dots, 1, m + 1, 2m + 1, \dots, 2, m + 2, 2m + 2, \dots, m - 1, 2m - 1, 3m - 1, \dots, nm - 1$. За исключением последнего элемента, который стоит на своём месте, все элементы перестановки a_i представляют собой $i \cdot m \pmod{nm - 1}$. Таким образом для каждого из элементов надо решить уравнение $i \cdot m^x = i \pmod{nm - 1}$. Решение этого уравнения зависит исключительно от наибольшего общего делителя i и $nm - 1$. Таким образом, перебирая все делители $nm - 1$, решаем уравнение для каждого из них, а дальше прибавляем к ответу количество таких чисел, делённое на ответ.

В этой задаче имеет смысл заметить, что все x являются делителями числа, являющегося решением в случае взимной простоты i и $nm - 1$, что позволяет существенно сократить перебор.

Задача C. Concave Polygon (высшая лига)

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

Дан простой многоугольник без самопересечений и самокасаний. Требуется разрезать его диагоналями на минимальное количество выпуклых многоугольников. В качестве вершин выпуклых многоугольников можно использовать только вершины исходного многоугольника. Каждая вершина исходного многоугольника должна встретиться в качестве вершины хотя бы одного выпуклого многоугольника. Никакие три вершины любого выпуклого многоугольника не должны лежать на одной прямой. В качестве сторон выпуклых многоугольников можно использовать стороны исходного многоугольника и его диагонали. Каждый из выпуклых многоугольников должен лежать внутри исходного многоугольника. Объединение всех выпуклых многоугольников должно составлять исходный многоугольник. Никакие два выпуклых многоугольника не должны иметь положительную площадь пересечения.

Формат входного файла

Первая строка содержит число N ($3 \leq N \leq 150$) — количество вершин исходного многоугольника. Каждая из последующих N строк содержит по два целых числа x_i и y_i ($-10\,000 \leq x_i, y_i \leq 10\,000$) — координаты i -ой вершины многоугольника в порядке обхода против часовой стрелки. Гарантируется, что многоугольник невырожденный.

Формат выходного файла

Одно число — минимальное количество выпуклых многоугольников, на которые можно разрезать исходный многоугольник.

Примеры

stdin	stdout
7 3 -1 3 1 2 1 3 3 0 0 3 -3 2 -1	3

Разбор задачи C. Concave Polygon

Задача решается с помощью динамического программирования. Пусть P_i — вершины многоугольника в порядке обхода против часовой стрелки. Пусть $d(i, j)$ — минимальное количество выпуклых многоугольников, на которые можно разбить многоугольник, состоящий из всех вершин исходного многоугольника с индексами от i до j в порядке обхода и диагонали $P_i P_j$, назовем его Q_{ij} . Заметим, что диагональ $P_i P_j$ должна лежать строго внутри исходного многоугольника.

Давайте зафиксируем некоторые i и j и найдем $d(i, j)$. Если полученный многоугольник уже является выпуклым, то $d(i, j) = 1$. В противном случае отрезок $P_i P_j$ должен будет принадлежать некоторому выпуклому многоугольнику. Переберем все выпуклые многоугольники, лежащие внутри Q_{ij} и содержащие отрезок $P_i P_j$. Воспользуемся методом динамического программирования для эффективного перебора всех возможных вариантов выбора главного выпуклого многоугольника.

Пусть $f(k, l)$ — минимальное количество многоугольников, на которые уже пришлось разбить Q_{ij} при условии, что главный выпуклый многоугольник, который содержит отрезок $P_i P_j$, мы частично построили, и он проходит через вершину k и l . То есть, отрезок $P_k P_l$ — его последняя на текущий момент сторона. Если отрезок $P_k P_l$ не принадлежит многоугольнику, то $f(k, l) = \infty$. Иначе найдем $f(k, l)$. Предыдущую сторону главного выпуклого многоугольника можно было провести через некоторую вершину t , и тогда:

$$f(k, l) = \min_{t \in T} \{f(t, k) + d(k, l)\}$$

Множество T — все вершины t в порядке обхода между i и k такие, что $[P_t P_k \times P_k P_l] > 0$, чтобы обеспечить выпуклость главного выпуклого многоугольника. $P_t P_k$ лежит внутри исходного многоугольника.

Все $f(i, k)$ нужно инициализировать тоже исходя из выпуклости главного многоугольника.

После чего:

$$d(i, j) = \min_{k, l} \{f(k, l) + d(k, j) + 1\}$$

При этом k и l выбираются так, чтобы отрезок $P_l P_j$ лежал внутри исходного многоугольника и главный выпуклый многоугольник действительно получался выпуклым. То есть, в частности $[P_k P_l \times P_l P_j] > 0$ и $[P_l P_j \times P_j P_i] > 0$.

Такое решение очевидно имеет сложность $O(N^5)$ и использует $O(N^2)$ памяти. Но если зафиксировать вершину k , а вершины l и t перебирать методом двух указателей по углу, то время работы можно сократить до $O(N^4)$.

Задача D. Dead Points (высшая лига)

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

На плоскости есть N точек. Точки бывают двух цветов: черные и белые. С точками происходит такой процесс: каждую итерацию любая точка, которая видит хотя бы одну точку противоположного цвета, умирает. Точка A видит точку B , если отрезок AB не содержит других живых точек множества. В течение одной итерации все необходимые точки умирают одновременно. Если не осталось живых точек какого-нибудь цвета, процесс заканчивается.

Определить, сколько итераций будет продолжаться процесс, и точки какого цвета выживут в результате.

Формат входного файла

В первой строке число N ($1 \leq N \leq 100\,000$) — количество точек на плоскости. Далее N строк по три целых числа в каждой: x_i , y_i и $color_i$ ($0 \leq x_i, y_i \leq 1\,000\,000$, $color_i \in \{0, 1\}$) — координаты и цвет i -ой точки. 0 означает белый, 1 — черный. Никакие две точки не имеют одинаковые координаты.

Формат выходного файла

Слово “Draw”, если в результате умерли все точки, “Black”, если черные точки выжили, или “White”, если выжили белые. В той же строке через пробел вывести количество итераций процесса.

Примеры

stdin	stdout
3 0 0 0 1 0 1 0 1 1	Draw 1
3 0 0 0 1 0 1 2 0 1	Black 1

Разбор задачи D. Dead Points

В задаче два принципиально различных случая. Когда все точки лежат на одной прямой и когда не все. Если все точки лежат на одной прямой, то за-

дача решается эмуляцией с помощью связных списков. Подробнее описано в разборе задачи Mortal Points из второго дивизиона.

Если не все точки лежат на одной прямой, то все точки как минимум одного из цветов умрут за один ход. Допустим, что это не так. Тогда у нас есть как минимум две белых и две чёрных точки на одной прямой. Если это не так, то очевидно, что все умрут за один ход. При этом помимо точек на этой прямой есть ещё одна точка не на прямой. Допустим, она белого цвета. Тогда черная точка на нашей прямой, которая выживет, должна быть закрыта другими точками чёрного цвета не на нашей прямой, но тогда надо будет закрыть выжившую точку белого цвета от этих чёрных. Несложно видеть, что этот процесс потребует бесконечного числа точек, что невозможно по условию.

Осталось определить, кто переживёт первый ход. Если белых или чёрных точек меньше \sqrt{N} — проверим за $O(N)$ для каждой точки, выживет ли она. Иначе выберем случайную точку и отсортируем все остальные по углу вокруг нее. Для каждой прямой, проходящей через эту точку, рассмотрим за $O(N)$, что там происходит. Также заметим, что любые два соседних луча, угол между которыми меньше π , могут беспрепятственно уничтожать точки друг на друге. Рассмотрим все такие соседние лучи. Выберем следующую точку, но так, чтобы она была другого цвета и при этом лежала на прямой с минимальным числом точек среди прямых, проходящих через центральную точку текущей итерации. Будем повторять этот процесс, пока новые точки не перестанут умирать. Если умерли все — ничья. Иначе — победитель очевиден.

Задача E. Equilateral Polygon

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

Требуется построить выпуклый многоугольник, длины сторон которого равны 1, а все углы при вершинах различны.

Формат входного файла

Единственное число N ($5 \leq N \leq 100$) — количество вершин многоугольника.

Формат выходного файла

Ровно N строк. i -ая строка должна содержать два вещественных числа x_i и y_i (рекомендуется с 16 знаками после запятой), которые являются координатами i -ой вершины многоугольника. x_i и y_i по модулю не должны превы-

шать 10^3 . Вершины можно выводить в порядке обхода по или против часовой стрелки. Длина каждой стороны не должна отличаться от 1 более чем на 10^{-8} . Любые два угла должны отличаться друг от друга как минимум на 10^{-4} радиан. Все углы должны быть в пределах $[10^{-4}, \pi - 10^{-4}]$.

Гарантируется, что хотя бы один многоугольник, удовлетворяющий условию, существует.

Примеры

stdin	stdout
5	1.000000000000000 1.000000000000000 2.000000000000000 1.000000000000000 2.3093247548365476 1.9509564638012140 1.5003804226047008 2.5388417139563542 0.6912907818637410 1.9511564638048255

Разбор задачи E. Equilateral Polygon

Основная идея — построить многоугольник из двух ломаных. Первая ломаная будет максимально близка к прямой. Это означает, что все углы при вершинах близки к π . У второй ломаной все углы почти равны между собой.

Первая ломаная состоит из $\frac{N}{3}$ вершин, тогда как вторая — из $N - \frac{N}{3} + 2$ вершин. Углы первой ломаной будут $\pi - \epsilon, \pi - 2\epsilon, \pi - 3\epsilon, \dots$. Углы второй ломаной будут $\omega - \epsilon, \omega - 2\epsilon, \omega - 3\epsilon, \dots$, где ω — это такой угол, при котором расстояние между первой и последней точками второй ломаной будет такое же, как и у первой. ω можно найти двоичным поиском.

Задача F. Flawless Numbers

Вход:	stdin
Выход:	stdout
Ограничение по времени:	5 с
Ограничение по памяти:	256 Мб

Число N называется безупречным, если $\frac{\sigma(N)}{N} = \frac{A}{B}$, где $\sigma(N)$ — сумма всех делителей числа N .

Для заданных A и B найдите все безупречные числа от 1 до 10^{14} включительно.

Формат входного файла

Два натуральных числа A и B ($1 \leq A, B \leq 100$, A — нечетное (внезапно), $2 \leq \frac{A}{B} \leq 5$).

Формат выходного файла

Первая строка должна содержать число K — количество безупречных чисел. Далее K строк, по одному безупречному числу в каждой. Каждое безупречное число должно встречаться ровно один раз. Числа можно выводить в любом порядке. Гарантируется, что K не превосходит 1000.

Примеры

stdin	stdout
5 2	3 24 91963648 10200236032

Разбор задачи F. Flawless Numbers

Решение — перебор с отсечением. Во-первых,

$$N = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_k^{e_k}$$

$$\sigma(N) = \frac{p_1^{e_1+1} - 1}{p_1 - 1} \cdot \frac{p_2^{e_2+1} - 1}{p_2 - 1} \cdot \dots \cdot \frac{p_k^{e_k+1} - 1}{p_k - 1}$$

Это позволяет добавлять делители к числу N по одному и пересчитывать значение $\sigma(N)$

Определим $S(i)$ — как некоторую оценку сверху максимума $\frac{\sigma(K)}{K}$ среди всех чисел $K \leq 10^{14}$, не содержащих первые i простых чисел в качестве простых делителей.

Запустим перебор $F(n, s, i)$ от нескольких параметров: n — текущее число, s — текущая сумма делителей, i — номер очередного простого числа, которое будем использовать в качестве делителя числа n . В качестве перехода в переборе выбирается степень k , в которой i -ое простое число будет входить в ответ: $F(n \cdot \text{Prime}[i]^k, s \cdot \frac{\text{Prime}[i]^{k+1} - 1}{\text{Prime}[i] - 1}, i + 1)$

Тогда работают несколько важных отсечений:

- Если $n \cdot Prime[i] > 10^{14}$ — выход
- Если $\frac{s}{n} > \frac{A}{B}$ — выход
- Если $S(i + 1) \cdot \frac{s}{n} < \frac{A}{B}$ — выход
- Если $s \cdot B$ содержит в качестве простых делителей такие простые числа, номера которых меньше i , и которые не содержатся в $n \cdot A$ — выход.

Эти отсечения позволяют перебору находить ответы для всех вариантов A и B в пределах предложенных ограничений.

Задача G. Grep

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

Дана строка S длины N из символов ‘а’, ‘б’ и ‘с’ и строка T длины N из символов ‘.’ и ‘*’. За одну операцию можно циклически сдвинуть строку S на один символ вправо или влево. Требуется за минимальное количество операций добиться того, чтобы на всех позициях, которые в строке T отмечены символом ‘*’, каждая из букв ‘а’, ‘б’ и ‘с’ побывала хотя бы один раз.

Формат входного файла

Первая строка содержит строку S , которая состоит только из символов ‘а’, ‘б’ и ‘с’, Каждый из символов ‘а’, ‘б’ и ‘с’ встречается в строке S хотя бы один раз. Вторая строка содержит строку T , состоящую только из символов ‘.’ и ‘*’. $3 \leq |S| = |T| \leq 1\,000\,000$.

Формат выходного файла

Одно число — минимальное количество операций сдвига, необходимых, чтобы на каждой позиции, отмеченной символом ‘*’, каждая из букв ‘а’, ‘б’ и ‘с’ побывала хотя бы раз.

Примеры

stdin	stdout
abacaba ...**..	3

Разбор задачи G. Grep

Для каждой звёздочки рассмотрим потенциально интересные сдвиги. Так как звёздочка уже стоит на одном из символов, то осталось покрыть ещё два. Для этого можно сдвинуть её налево, пока не встретятся оба; налево до ближнего символа и потом направо до оставшегося; направо, пока не встретятся оба. Нам надо найти такую пару сдвигов налево и направо, которые бы удовлетворяли одновременно всем звёздочкам.

Рассмотрим максимальный сдвиг налево. Он удовлетворяет всем звёздочкам. Начнём уменьшать сдвиг налево. Возможно, после какого-то уменьшения, чтобы удовлетворить какую-то звёздочку нам понадобится увеличить сдвиг направо. Заметим, что сдвиг направо никогда не будет уменьшаться. Рассмотрим все возможные сдвиги налево, для каждого из них найдём соответствующий минимальный сдвиг направо и из них выберем оптимальную пару. Рассматривая какую-нибудь пару сдвигов (l, r) , надо учитывать, что какой-то из двух сдвигов придётся осуществить дважды (вернуться). Поэтому стоимость пары — это $2 \min(l, r) + \max(l, r)$.

При реализации алгоритма для каждой точки можно генерировать пару событий: сдвиг налево, при котором появляется сдвиг направо и сдвиг налево, при котором сдвиг направо увеличивается. Отсортировав эти события по убыванию сдвига налево легко перебрать и обработать все интересные для нас сдвиги налево.

Задача H. Hash It!

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

Изначально есть пустая строка S . Над строкой требуется производить операции двух типов:

- Добавить к строке S справа символ c .
- Удалить из строки S один символ справа.

После каждой операции требуется определить количество различных непустых подстрок строки S .

Формат входного файла

Одна строка Q ($1 \leq |Q| \leq 100\,000$), означающая последовательность операций со строкой S . Если i -ый символ строки Q является строчной буквой английского алфавита, тогда в качестве i -ой операции нужно добавить эту букву

в конец строки S , если i -ый символ строки Q является символом ‘–’, тогда в качестве i -ой операции нужно удалить из строки S один символ справа. Никакие другие символы в строке Q не встречаются. Гарантируется, что перед любой операцией удаления строка S не является пустой.

Формат выходного файла

Вывести ровно $|Q|$ строк. i -ая строка должна содержать одно число — количество различных подстрок строки S после применения первых i операций.

Примеры

stdin	stdout
aba–caba	1 3 5 3 6 9 12 17

Разбор задачи H. Hash It!

Пусть символ добавляется не в конец, а в начало строки. Удаляется тоже. Очевидно, что при этом количество различных подстрок не изменится. Количество различных подстрок в строке равно сумме длин суффиксов минус сумма длин всех LCP. Будем хранить все текущие суффиксы в сэте, отсортированными лексикографически. Сравнивать суффиксы будем за $O(\log(N))$ с помощью хешей и двоичного поиска. Будем поддерживать суммарную длину всех LCP. Вставляя суффикс, вычтем длину LCP тех суффиксов, между которыми мы вставляем новый, и добавим длины LCP нового суффикса с соседними лексикографически. Таким образом мы пересчитали суммарную длину LCP всех суффиксов и с помощью нее число различных подстрок. При удалении символа удалим соответствующий суффикс из сэта, и пересчитаем соответствующие LCP аналогичным образом.

Итоговая сложность $O(N \log^2 N)$.

Задача I. Interactive Problem 2

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	10 Мб

Обратите внимание! Эта задача имеет специальное ограничение по памяти.

Дано корневое бинарное дерево. Все вершины дерева пронумерованы числами от 1 до N . У каждой вершины может быть либо 2 сына, либо 0. Требуется обойти дерево поиском в ширину или поиском в глубину — по вашему усмотрению, и вывести номера вершин в порядке выбранного вами обхода, а также сумму расстояний от корня до всех вершин. Корень всегда имеет номер 1.

Если вы выводите вершины в порядке обхода поиска в ширину, то выводить номера вершин в пределах одного слоя следует слева направо. Если вы выводите вершины в порядке обхода поиска в глубину, то номер вершины следует выводить в момент первого посещения, а при обходе сначала идти в левого сына, потом в правого.

Более формально: поставим каждой вершине v_i в соответствие строку S_i , j -ым символом которой является ' l ', если j -ое ребро на пути из корня в вершину v_i ведёт в левого сына и ' r ', если в правого. Количество символов в строке S_i равно расстоянию от вершины до корня.

При обходе поиском в ширину вершина v_i должна быть посещена раньше вершины v_j тогда и только тогда, когда:

- S_i короче S_j , если S_i и S_j имеют разную длину
- S_i лексикографически меньше S_j , если S_i и S_j имеют одинаковую длину

При обходе поиском в глубину вершина v_i должна быть посещена раньше вершины v_j тогда и только тогда, когда S_i лексикографически меньше S_j .

Формат входного файла

Первая строка входного файла содержит число N ($1 \leq N \leq 1\,000\,000$) — количество вершин в дереве. Каждая из следующих N строк содержит пару чисел l_i, r_i ($1 \leq l_i, r_i \leq N$) — номера левого и правого сыновей вершины i , или два нуля, если вершина i является листом.

Формат выходного файла

В первой строке выведите “DFS”, если вы обходите дерево поиском в глубину, или “BFS”, если поиском в ширину. В последующих N строках выведите

номера вершин в порядке выбранного вами обхода. В последней строке выведите сумму расстояний от корня до всех вершин.

Примеры

stdin	stdout
5	DFS
3 4	1
0 0	3
0 0	4
2 5	2
0 0	5
	6

Разбор задачи I. Interactive Problem 2

Будем хранить вершины дерева в виде структуры с двумя указателями на левого и правого сына. Заметим, что при таком задании дерева у нас недостаточно памяти ни для стека, ни для очереди, для того что бы воспользоваться каким-нибудь стандартным алгоритмом обхода в глубину или в ширину. Еще заметим, что нам вовсе не надо сохранять дерево. То есть, мы можем повторно использовать те вершины, которые мы уже просмотрели.

Рассмотрим обход в глубину. Заведём указатель на текущую вершину. Будем передвигать этот указатель пока не обойдем всё дерево. Если мы только пришли в вершину, то нам надо идти в левого сына. Если мы только что вернулись из левого сына, то нам надо в правого. Если мы только что вернулись из правого сына, то нам надо идти в предка. Но у нас, во-первых, недостаточно памяти, чтобы хранить ссылку на предка, и у нас есть сложности с определением того, каким было последнее действие. Эти проблемы можно решить так: когда мы только входим в вершину, мы знаем из какой вершины мы пришли, то есть, мы знаем предка. Нам этого предка необходимо где-то сохранить. Заметим, что мы на следующем шаге уйдём в левого сына и больше никогда на него не посмотрим — то запомним во временной переменной номер левого сына, и вместо него запишем предка.

Теперь надо научиться определять, откуда мы пришли. Для этого будем хранить направление движения. Оно нам также понадобится, чтобы найти расстояние до корня. Пусть переменная *dir* будет равна 1, если мы движемся вниз, и -1 иначе. Если мы пришли в вершину двигаясь вниз, то, очевидно, что нам надо переписать указатель на предка и пойти в левого сына. Если мы пришли двигаясь вверх, то нам надо либо пойти в правого сына, если мы его ещё не посещали, либо в предка, если мы вернулись из правого сына. Для этого

перед тем как идти в правого сына, запомним его номер и обнулим ссылку. Теперь, посмотрев на правого сына, мы знаем, что мы его ещё не посещали, если указатель на него не обнулён, и знаем, что мы его посетили и нам надо возвращаться (через ссылку на предка, хранящуюся на данный момент в левом сыне), если ссылка на правого сына обнулена.

Для обхода в ширину заметим следующее. Так как дерево у нас бинарное, то максимальное число вершин, которые могут быть в очереди, не превышает число уже рассмотренных вершин плюс ещё одна вершина. Значит, в каждый момент времени, если мы освободим память, которую у нас занимали уже рассмотренные вершины, то нам хватит памяти на очередь. Будем аккуратно выделять и освобождать память и напишем обычный обход в ширину. Единственная проблема — недостаточно памяти для хранения расстояния до каждой вершины. Чтобы ее решить, будем поддерживать номер текущего слоя, который и будет расстоянием до корня. Для этого для каждого слоя будем хранить номер последней вершины в этом слое. И будем хранить номер последней добавленной в очередь вершины. Если текущая рассматриваемая вершина является последней в слое (мы это знаем, так как храним номер последней вершины в слое), то рассмотрев её отметим последнюю добавленную в очередь вершину как последнюю в следующем слое. И так как очевидно, что мы как раз переходим на следующий слой, обновим указатель на конец текущего слоя.

Задача J. Jolly Dolls

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

Матрешки — деревянные куклы, вложенные одна в другую. Каждая кукла имеет собственный внешний объем out_i и объем in_i пустого места внутри нее. Одну куклу можно вложить в другую, если внешний объем первой меньше (в этот момент вы спрашиваете себя “меньше???”). Да, вам не показалось, именно меньше) объема пустого места внутри второй. В каждую куклу можно непосредственно вложить не более одной куклы. Но эта вложенная кукла может содержать в себе другую куклу и так далее. То есть, если две куклы фактически находятся внутри третьей, то одна из них находится внутри другой.

Матрешки вложены друг в друга оптимально, если суммарный объем пустого пространства внутри всех матрешек минимален. Требуется найти количество способов оптимально вложить матрешки друг в друга.

Формат входного файла

Первая строка содержит число N ($1 \leq N \leq 100\,000$) — количество матрешек в наборе. i -ая из следующих N строк содержит два целых числа out_i и in_i ($1 \leq in_i < out_i \leq 1\,000\,000\,000$) — внешний объем и объем пустого места внутри i -ой матрешки.

Формат выходного файла

Количество способов оптимального расположения матрешек по модулю 1 000 000 007.

Примеры

stdin	stdout
3	1
5 4	
4 2	
3 2	

Разбор задачи J. Jolly Dolls

Сначала найдем некоторый способ оптимального расположения матрешек. Отсортируем матрешки по убыванию внешнего объема. Будем поддерживать множество S всех матрешек, в которых на текущий момент нет других матрешек. При рассмотрении очередной матрешки есть два варианта: либо ее можно поместить в некоторую матрешку из множества S , либо не помещать никуда. Если текущая матрешка вкладывается в другую, то эта другая удаляется из множества S . После рассмотрения текущей матрешки она добавляется в множество S . Заметим, что не имеет смысла не помещать матрешку в другую матрешку. С другой стороны нам неважно в какую из допустимых матрешек мы поместим текущую. Это так, потому что все следующие матрешки по размеру меньше или равны текущей и поэтому количество матрешек, допустимых для вставки следующих матрешек, не изменится в зависимости от того, куда мы вставим текущую.

При вставке текущей матрешки нужно понять, сколько на текущий момент существует матрешек в множестве S таких, что в каждую из них можно вложить текущую матрешку. Это можно сделать с помощью дерева отрезков.

Стоит отметить, что все матрешки с одинаковым внешним объемом следует рассматривать одновременно для того, чтобы учесть случай, когда одинаковых матрешек больше, чем тех в множестве S , в которые их можно вложить. В такой ситуации мы можем произвольным образом поделить одинаковые матрешки на те, которые будут вложены в другие и те, которые не будут никуда

вложены. Результат при этом дополнительно умножается на число сочетаний из количества одинаковых матрешек по количеству доступных мест в S .

Задача K. Block Shuffling (юниорская лига)

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

Есть тождественная перестановка из 2^p элементов $(1, 2, 3, \dots, 2^p)$. Требуется произвести с ней последовательность операций такого вида:

Разбить перестановку на непрерывные непересекающиеся блоки длиной 2^k . То есть, элементы на позициях $(1, \dots, 2^k)$ формируют первый блок, элементы на позициях $(2^k + 1, \dots, 2 \cdot 2^k)$ — второй блок, элементы $(2 \cdot 2^k + 1, \dots, 3 \cdot 2^k)$ — третий блок, и так далее. После этого соседние блоки поменять местами. То есть, первый блок меняется со вторым, третий с четвертым и так далее.

Дано множество позиций, требуется сказать, какие элементы перестановки находятся на этих позициях после выполнения всех операций.

Формат входного файла

Первая строка содержит три числа N, K и Q ($N = 2^p, 0 \leq p \leq 21, 1 \leq K, Q \leq 100\,000$) — количество элементов перестановки, количество операций над перестановкой и количество интересующих позиций.

Далее следуют K строк по одному числу k_i в каждой ($0 \leq k_i < p$), означающих, что нужно менять соседние блоки по 2^{k_i} элементов.

Далее следуют Q строк по одному числу q_i в каждой ($1 \leq q_i \leq N$) — номера интересующих позиций.

Формат выходного файла

Вывести Q строк, в i -ой из которых номер элемента перестановки на позиции q_i .

Примеры

stdin	stdout
8 3 3	8
0	6
1	2
2	
1	
3	
7	

Разбор задачи K. Block Shuffling

Заметим, что когда мы меняем местами все соседние блоки длины 2^i , i -ый бит каждого элемента перестановки инвертируется, при этом остальные биты не меняются (это утверждение справедливо, если элементы нумеруются с 0, а не с 1 как в условии). То есть, достаточно посчитать, сколько раз каждый бит всех элементов менялся на противоположный. После чего для каждого элемента понятно, какие значения имеют его биты. Проще всего это сделать применяя операцию побитового исключающего “или” индекса элемента (при нумерации с 0) и всех измененных битов.

Задача L. Digit Permutation (юниорская лига)

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

Найти число в системе счисления по основанию B , удовлетворяющее следующим ограничениям:

- Каждая цифра от 1 до $B - 1$ встречается ровно один раз (цифра 0 не встречается).
- Для любого i : $1 \leq i \leq B - 1$, если взять первые i цифр этого числа, не меняя порядок, то полученное число делится на i .

Формат входного файла

Число B ($2 \leq B \leq 36$) — основание системы счисления.

Формат выходного файла

Если требуемого числа не существует, вывести “No”, иначе вывести одно число в системе счисления по основанию B . Цифры от 0 до 9 задаются символами ‘0’ – ‘9’, Цифры от 10 до 35 задаются символами ‘A’ – ‘Z’. Если решений несколько выведите любое из них.

Примеры

stdin	stdout
10	381654729
18	No

Разбор задачи L. Digit Permutation

Для нечётных B ответ всегда “No”, так как по признаку делимости, чтобы в B -ичной системе счисления число делилось на $B - 1$, необходимо, чтобы сумма его цифр делилась на $B - 1$, что невозможно для арифметической прогрессии нечётной длины с первым элементом 1 и разностью 1.

Для чётных B найдём ответ перебором. Заметим, что на чётных позициях у нас могут быть только чётные цифры, так как для того, чтобы число делилось на чётное (у нас чётная позиция и именно на неё мы и делим), число само должно быть чётным. Соответственно на нечётных позициях у нас могут быть только нечётные цифры, так как чётных и нечетных цифр, равно как и позиций, у нас поровну. Это отсечение позволяет рекурсивному перебору уложиться в таймлимит при данных ограничениях.

Задача M. Mortal Points (юниорская лига)

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

На прямой есть N точек. Точки бывают двух цветов: черные и белые. С точками происходит такой процесс: каждую итерацию любая точка, которая видит хотя бы одну точку противоположного цвета, умирает. Точка A видит точку B , если отрезок AB не содержит других живых точек множества. В течение одной итерации все точки умирают одновременно. Если не осталось живых точек какого-нибудь цвета, процесс заканчивается.

Определить, сколько итераций будет продолжаться процесс, и точки какого цвета выживут в результате.

Формат входного файла

В первой строке число N ($1 \leq N \leq 100\,000$) — количество точек на прямой. Далее N строк по два целых числа в каждой: x_i и $color_i$ ($0 \leq x_i \leq 1\,000\,000\,000$, $color_i \in \{0, 1\}$) — координата и цвет i -ой точки. 0 означает белый, 1 — черный. Никакие две точки не имеют одинаковые координаты.

Формат выходного файла

Слово “Draw”, если в результате умерли все точки, “Black”, если черные точки выжили, или “White”, если выжили белые. В той же строке через пробел вывести количество итераций процесса.

Примеры

stdin	stdout
3 0 0 1 1 2 1	Black 1

Разбор задачи M. Mortal Points

Будем эмулировать процесс. Будем хранить точки на прямой в виде списка. У каждой точки будет указатель на соседнюю слева и справа. Отдельно будем поддерживать список указателей на точки, являющиеся крайними на отрезках, состоящих из точек одного цвета, так как только эти точки могут повлиять на ситуацию, а именно на своих соседей.

Будем повторять следующий процесс, пока у нас не умрут все точки как минимум одного цвета: просмотрим все указатели, указывающие на вершины, находящиеся на краях отрезков из точек одного цвета. Посмотрим на их соседей. Если есть сосед другого цвета, удалим его и текущую вершину из списка, сдвинув указатели на края отрезков соответствующим образом. Если обновленные указатели уже есть в множестве — удалим их. Если среди соседей удаляемых вершин нет вершин того же цвета — тоже удалим указатели. Если оба соседа рассматриваемой вершины того же цвета, что и вершина — удалим указатель. Заметим, что при каждом просмотре указателя уменьшается либо число указателей либо число вершин. Отсюда следует, что мы сделаем $O(N)$ операций.

Задача N. Non-convex Polygon (юниорская лига)

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

Дан простой многоугольник без самопересечений и самокасаний. Требуется разрезать его диагоналями на минимальное количество выпуклых многоугольников. В качестве вершин выпуклых многоугольников можно использовать только вершины исходного многоугольника. Каждая вершина исходного многоугольника должна встретиться в качестве хотя бы одного выпуклого многоугольника. Никакие три вершины любого выпуклого многоугольника не должны лежать на одной прямой. В качестве сторон выпуклых многоугольников можно использовать стороны исходного многоугольника и его диагонали. Каждый из выпуклых многоугольников должен лежать внутри исходного многоугольника. Объединение всех выпуклых многоугольников должно составлять исходный многоугольник. Никакие два выпуклых многоугольника не должны иметь положительную площадь пересечения.

Формат входного файла

Первая строка содержит число N ($3 \leq N \leq 15$) — количество вершин исходного многоугольника. Каждая из последующих N строк содержит по два целых числа x_i и y_i ($-10\,000 \leq x_i, y_i \leq 10\,000$) — координаты i -ой вершины многоугольника в порядке обхода против часовой стрелки. Гарантируется, что многоугольник невырожденный.

Формат выходного файла

Одно число — минимальное количество выпуклых многоугольников, на которые можно разрезать исходный многоугольник.

Примеры

stdin	stdout
<pre>7 3 -1 3 1 2 1 3 3 0 0 3 -3 2 -1</pre>	3

Разбор задачи N. Non-convex Polygon

Задача решается методом динамического программирования, аналогично задаче С первого дивизиона. Но ограничения позволяют перебирать в явном виде все подмножества вершин в качестве главного выпуклого многоугольника. Сложность $O(2^N N^2)$.

День девятый (23.02.2014 г.) Конкурс Виталия Неспирного

Об авторе...

Неспирный Виталий Николаевич, кандидат физико-математических наук, старший научный сотрудник отдела технической механики ИПММ НАН Украины, 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 ACM-олимпиаде по программированию в составе команды математического факультета ДонНУ (Винница, 2001).
- Подготовил 3 призеров международных школьных олимпиад.
- Тренер команды DonNU United, занявшей 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. Построение пересечения полу- плоскостей

Построение общей области N полуплоскостей эквивалентно поиску области решений системы из N линейных неравенств (ограничений) типа

$$a_i x + b_i y + c_i \leq 0, \quad i = \overline{1, n}. \quad (*)$$

Любое решение системы $(*)$ называется допустимым решением, а все их множество называется допустимой областью.

Итак, у нас имеется N полуплоскостей H_1, H_2, \dots, H_N . Необходимо найти их пересечение $H = H_1 \cap H_2 \cap \dots \cap H_N$.

Существует простой квадратичный алгоритм для построения пересечения N полуплоскостей. Предположим, что уже известно пересечение первых i полуплоскостей. Это некая выпуклая многоугольная область, ограниченная не более чем i сторонами, хотя и необязательно конечная. Пересечение этой области с очередной полуплоскостью есть не что иное, как ее разрезание прямой линией и сохранение правого или левого куска. Это можно проделать очевидным способом за время $O(i)$. Суммарная работа потребует $O(N^2)$ времени.

Рассмотрим вопрос о возможном улучшении алгоритма с помощью метода “разделяй и властвуй”. Введем следующее обозначение:

$$H(i, j) = H_i \cap \dots \cap H_j, \quad i \leq j.$$

Поскольку операция пересечения ассоциативна, то для любых $i \leq k < j$ верно равенство

$$H(i, j) = H(i, k) \cap H(k+1, j).$$

Так, например,

$$H(1, N) = H(1, N/2) \cap H(N/2 + 1, N).$$

Здесь $H(1, N/2)$ является пересечением $N/2$ полуплоскостей и, следовательно, выпуклой многоугольной областью не более чем с $N/2$ сторонами. То же самое относится и к $H(N/2 + 1, N)$.

Таким образом приходим к следующему алгоритму:

Data: N полуплоскостей, заданных ориентированными прямолинейными отрезками

Result: Их пересечение, выпуклая многоугольная область

begin

Разбить заданные полуплоскости на два подмножества приблизительно равной мощности

Рекурсивно построить пересечение полуплоскостей для каждого из этих подмножеств

Построить пересечение решений подзадач, являющихся двумя выпуклыми многоугольными областями

end

Обозначить через $T(N)$ время, используемое этим алгоритмом для построения пересечения N полуплоскостей. Если нам удастся построить пересечение двух выпуклых многоугольных областей, каждая из которых содержит не более чем $N/2$ сторон, за время $O(N)$, то будет иметь место рекуррентное соотношение:

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

Это хорошо известное соотношение, справедливое для многих задач, решаемых с помощью метода “разделяй и властвуй” и оно имеет решение:

$$T(N) = O(N \log N).$$

Это решение имеет наилучшую возможную оценку для данной задачи. Предположим обратное: пусть существует алгоритм с более оптимальной асимптотической оценкой. Покажем, как с помощью этого алгоритма решить задачу сортировки N действительных чисел x_1, \dots, x_N . Возьмем в качестве H_i полуплоскость, содержащую начало координат и определяемую касательной к параболе $y = x^2$ в точке с координатами (x_i, x_i^2) , то есть прямой $y = 2x_i x - x_i^2$. Пересечением таких полуплоскостей является выпуклая многоугольная область, последовательные ребра которой упорядочены по углу наклона. Если построить указанную область, то можно будет найти за $O(N)$ минимальное значение x_1 и начиная от прямой, соответствующей данной точке, получить (путем обхода по контуру многоугольной области) последовательность $\{x_i\}$ в отсортированном порядке. Однако, как известно, не существует алгоритма сортировки вещественных чисел, работающего быстрее, чем за время порядка $N \log N$. Следовательно, предположение о существовании алгоритма с лучшей асимптотикой было неверным.

Рассмотрим теперь задачу построения пересечения двух выпуклых многоугольных областей.

Построение пересечения выпуклых многоугольников

Пусть даны два выпуклых многоугольника: P с L вершинами и Q с M вершинами. Необходимо построить их пересечение.

Под “многоугольником” будем понимать его границу и внутренность; цикл из ребер многоугольника будет явно называться его границей.

Существует несколько решений этой задачи, которые имеют различную вычислительную сложность алгоритма и сложность для реализации.

Ясно, что пересечением двух выпуклых многоугольников является опять же выпуклый многоугольник. Можно показать, что он будет иметь не более, чем $L+M$ вершин. Действительно, результирующий многоугольник является пересечением полуплоскостей, определяемых $L + M$ сторонами данных многоугольников. Каждое ребро результирующего многоугольника будет определяться одной из этих полуплоскостей и, в силу выпуклости, не может быть двух ребер соответствующих одной и той же полуплоскости. Следовательно, ребер (а значит и вершин) не может быть больше, чем $L + M$.

Граница $P \cap Q$ состоит из чередующихся цепей вершин этих двух многоугольников, разделенных точками пересечения их границ. Поэтому, если мы найдем все ее вершины в порядке обхода, то мы решим задачу.

Очевидный метод формирования пересечения состоит в обходе границы одного из многоугольников ребро за ребром, в процессе которого просто определяются все точки пересечения с границей второго многоугольника (не более двух на каждое ребро) и попутно ведется список точек пересечения и вершин. Это потребует затрат времени $O(LM)$, поскольку будут проверены пересечения каждого ребра P с каждым ребром Q . Естественно для сокращения вычислительной работы попытаться использовать факт выпуклости многоугольников.

Один из подходов заключается в разбиении плоскости на области, в каждой из которых пересечение этих двух многоугольников можно вычислить легко. В данном случае будет адекватным простейшее разбиение плоскости, индуцированное пучком лучей. Выберем произвольную точку O на плоскости (эту точку можно взять на бесконечности). Из O проводим лучи через все вершины многоугольников P и Q . Эти лучи разбивают плоскость на секторы. Мы хотим упорядочить этот набор лучей по углу вокруг точки O . Это можно легко сделать. На самом деле, если O лежит внутри многоугольника, скажем, P , то лучи к вершинам P , очевидно, упорядочиваются относительно O так же, как и вершины, порядок которых задан. Если же O лежит вне P , то вершины P образуют две цепи (обе они заключены между теми двумя вершинами P , которых касаются опорные прямые из O), а для каждой цепи соответствующие ей лучи из O отсортированы по углу. Поэтому в любом случае эти секторы, определяемые вершинами P , упорядочиваются по углу за время, линейно зависящее

от числа вершин. Если эти две последовательности (одна для P и одна для Q) известны, то их можно объединить за линейное время. Ключевым является наблюдение, что пересечение каждого сектора с выпуклым многоугольником образует четырехугольник. Поэтому внутри каждого сектора пересечением P и Q будет пересечение двух четырехугольников, которое можно найти за константное время. Результирующие куски можно собрать воедино за один обход секторов, на который понадобится линейное время. А затем потребуется очищающий просмотр для удаления ложных вершин, которые возникают на границах между секторами.

Таким образом, задача будет решена за время $O(L + M)$.

Другой подход является усовершенствованием того метода, который был описан в самом начале. Его основная идея следующая. Предположим, что $\partial P \cap \partial Q \neq \emptyset$, и рассмотрим многоугольник $P^* = P \cap Q$. Его границей является чередующаяся последовательность участков границ P и Q . Если одним из таких участков является кусок границы, скажем, P , то некий участок границы Q окружает ее во внешней области P^* . Благодаря выпуклости обоих многоугольников, в каждой паре таких смежных участков выделяются внешняя и внутренняя цепи, которые в совокупности называются “серпом”. Серп ограничен парой точек пересечения, именуемых начальной и конечной в соответствии с единой ориентацией границ P и Q . Говорят, что вершина (из P или Q) принадлежит серпу, если выполнено одно из двух условий

1. она лежит между начальной и конечной точками пересечения (на любой цепи);
2. является концом ребра внутренней цепи, содержащего конечную точку серпа.

Заметим, что существуют вершины, принадлежащие двум серпам.

Для создания механизма продвижения положим, что (p_1, p_2, \dots, p_l) и (q_1, q_2, \dots, q_m) — это списки вершин P и Q соответственно, перечисленных при обходе их против часовой стрелки (так что многоугольник считается расположенным “слева” от своей границы). Предположим, что продвижение осуществляется одновременно по обеим границам и что p_i и q_j — текущие вершины заданных многоугольников. Кроме того, текущие ребра оканчиваются в p_i и q_j . Если текущие ребра пересекаются, то мы выводим их пересечение в ответ. При этом внешняя и внутренняя цепи меняются ролями. Дальнейшее продвижение осуществляется с тем расчетом, чтобы не двигаться по той границе (P или Q), текущее ребро которой еще может содержать необнаруженную точку пересечения.

Общую схему алгоритма можно записать таким образом (здесь предполагается, что $p_0 = p_l$ и $q_0 = q_m$ непосредственно предшествуют p_1 и q_1 соответственно):

```
begin
     $i := j := k := 1$ 
    while  $k < 2(L + M)$  do
        if отрезки  $p_{i-1}p_i$  и  $q_{j-1}q_j$  пересекаются then
            | вывод пересечения
        end
        ДВИЖЕНИЕ
         $k = k + 1$ 
    end
    if не найдено пересечений then
        if  $p_i \in Q$  then
            |  $P \subseteq Q$ 
        end
        else if  $q_j \in P$  then
            |  $P \subseteq Q$ 
        end
        else
            |  $P \cap Q = \emptyset$ 
        end
    end
end
```

Алгоритм проходит одновременно по границам многоугольников P и Q , выделяя внутреннюю и внешнюю цепь для каждого серпа и находя все точки пересечения. На каждом шаге мы продвигаемся к следующему против часовой стрелки ребру многоугольника P либо Q . Если при таком движении за $L + M$ шагов не будет обнаружено точек пересечения, то границы многоугольников не пересекаются. И тогда либо один многоугольник содержится внутри другого, либо они не имеют пересечения. Если же будет найдено пересечение границ, то еще не более чем $L + M$ шагов понадобится для построения многоугольника P^* .

Еще один достаточно простой алгоритм приведен в работе [Toussaint G. T. (1985) A simple linear algorithm for intersecting convex polygons]. В отличии от предыдущих алгоритмов, он является комбинацией известных простых процедур для вычисления выпуклой оболочки и триангуляции многоугольников. Сначала строится выпуклая оболочка объединения многоугольников, затем определяются “мосты”, т.е. такие ребра в выпуклой оболочке, концы которых принадлежат разным многоугольникам. Каждому мосту соответствует точка пересечения границ P и Q , которая может быть найдена с помощью триангуляции многоугольника, ограниченного этим мостом и частями границ P и Q . Наконец, остается “соединить” найденные точки пересечения соответствую-

щими цепями границ Р либо Q. Последний алгоритм более прост для реализации, поскольку требует рассмотрения меньшего числа крайних случаев. Однако на практике работает несколько дольше рассмотренного выше алгоритма.

Задачи и разборы

Задача А. Максимальная разность (высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Вася выбирает два натуральных числа, стандартная запись каждого из которых в системе счисления с основанием b состоит из N цифр, и сумма цифр в этих записях одинаковы. Вася хочет сделать выбор таким образом, чтобы разность между числами была максимально возможной. Помогите Васе найти эту разность.

Ограничения

$$2 \leq b \leq 36, 1 \leq N \leq 10^5.$$

Формат входного файла

Единственная строка содержит два целых числа b и N .

Формат выходного файла

Выведите единственное целое число — максимальную разность между N -значными числами в системе счисления с основанием b с одинаковой суммой цифр. Результат должен быть представлен в стандартной записи в системе с основанием b . Для цифр, больших 9, используйте большие латинские буквы: A, B, C, \dots, Z .

Пример

<code>stdin</code>	<code>stdout</code>
10 2	72
16 3	E01

Задача В. Максимальная разность (юниорская лига)

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Вася выбирает два натуральных N -значных числа с одинаковой суммой цифр. Он хочет сделать это таким образом, чтобы разность между ними была максимально возможной. Помогите Васе сделать такой выбор.

Ограничения

$$1 \leq N \leq 10.$$

Формат входного файла

Единственная строка содержит целое число N .

Формат выходного файла

Выведите единственное целое число — максимальную разность между N -значными числами с одинаковой суммой цифр.

Пример

stdin	stdout
2	72

Разбор задачи В. Максимальная разность

Переберем все возможные суммы цифр S от 1 до $(b - 1) \times N$. Очевидно, чтобы разность двух чисел с такой суммой была максимальной, необходимо выбрать первое число максимально возможным, а второе — минимально возможным. Для построения первого числа будем распределять сумму, начиная со старшего разряда к младшим. Пока остаток суммы больше или равен $b - 1$ делаем текущий разряд равным $b - 1$ и переходим дальше, в противном случае полагаем текущий разряд равным остатку суммы. То же самое делаем и для построения второго числа за исключением того, что начинать следует с младших разрядов, двигаясь к старшим, и одну единицу нужно сразу положить в старший разряд. Построение этих чисел и вычисление их разности требует порядка $O(N)$ операций. Поэтому общая сложность алгоритма будет иметь вид $O(bN^2)$, что вполне достаточно для решения варианта Junior.

В варианте High требуется заметить, что переход от суммы S к $S + 1$ отражается в увеличении на единицу одного разряда в первом числе и одного разряда во втором (самые первые с соответствующей стороны разряды, отличные

от $b - 1$). Для отметки этих разрядов используются два указателя, каждый из которых продвигается на одну позицию, если на очередном шаге мы получаем в соответствующем разряде значение $b - 1$. Кроме того, разность можно не пересчитывать, достаточно знать, что она будет увеличиваться до тех пор, пока изменяющийся разряд в первом числе будет более значимым (т.е. старше), чем разряд во втором числе. Лишь в самом конце потребуется посчитать разность построенных чисел. Таким образом, имеем алгоритм со сложностью $O(bN)$, что будет уже достаточно и для варианта High.

Однако может быть построен алгоритм со сложностью $O(N)$, который вычисляет непосредственно разность. Для этого заметим, что разность цифр в одном и том же разряде двух чисел может быть от $-(b - 1)$ до $b - 1$. Будем поддерживать баланс суммы dS , определяемый как разность между суммами цифр первого и второго числа. Изначально она равна 0. На каждом шаге мы должны выбирать в разряд ответа такую цифру от $-(b - 1)$ до $b - 1$, чтобы получившийся баланс можно было компенсировать за счет оставшихся разрядов. Нетрудно видеть, что это число равно $\min((b - 1) \times k - dS, b - 1)$, где k — количество оставшихся разрядов. Исключение составляется самый старший разряд в котором разность цифр может быть в диапазоне от $-(b - 2)$ до $b - 2$. В конце мы получим все “цифры” разности, но число необходимо будет нормализовать, поскольку некоторые “цифры” могут оказаться отрицательными.

Следует обратить особое внимание на случаи $b = 2$ и $b = 3$, в которых один (или иногда два) старших разряда разности могут обратиться в 0.

Задача С. Построение многоугольника (высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Дана последовательность из N целых чисел a_1, a_2, \dots, a_N . Требуется построить выпуклый многоугольник с указанными длинами сторон, никакие три вершины которого не лежат на одной прямой.

Ограничения

Все заданные значения являются целыми и не превосходят 10^5 по абсолютной величине, $N \geq 3$.

Формат входного файла

Первая строка содержит целое число N . Во второй строке записаны числа

a_1, a_2, \dots, a_N .

Формат выходного файла

Выведите N строк. В каждой строке должны находиться декартовы координаты соответствующей вершины треугольника. Расстояние между вершинами с номерами i и $i + 1$ должно быть равно a_i , между вершинами с номерами N и 1 – значению a_N . Все равенства должны выполняться с точностью не менее 10^{-5} . Все координаты не должны превышать $2 \cdot 10^9$ по абсолютной величине. Если построить многоугольник с требуемыми свойствами невозможно, выведите одну строку “**Impossible**”.

Пример

stdin	stdout
3 3 4 5	0.0000000 3.0000000 0.0000000 0.0000000 4.0000000 0.0000000
3 1 2 3	Impossible
5 2 5 13 10 10	0.0000000 0.0000000 2.0000000 0.0000000 5.0000000 4.0000000 0.0000000 16.0000000 -6.0000000 8.0000000

Задача D. Построение треугольника (юниорская лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Даны три целых числа a , b и c . Требуется построить невырожденный треугольник с указанными длинами сторон.

Ограничения

Значение a , b и c не превосходят 10^5 по абсолютной величине.

Формат входного файла

Единственная строка содержит три целых числа a , b и c .

Формат выходного файла

Выведите 3 строки. В каждой строке должны находиться декартовы координаты соответствующей вершины треугольника. Расстояние между первой и второй вершиной должно быть равно a , между второй и третьей – b , между первой и третьей – c . Все равенства должны выполняться с точностью до 10^{-5} . Все координаты не должны превышать 10^6 по абсолютной величине. Если построить невырожденный треугольник с указанными сторонами невозможно, выведите одну строку “**Impossible**”.

Пример

stdin	stdout
3 4 5	0.0000000 3.0000000 0.0000000 0.0000000 4.0000000 0.0000000
1 2 3	Impossible

Разбор задачи D. Построение многоугольника

Рассмотрим задачу о возможности построения треугольника по заданным сторонам a, b, c . Известно следующее утверждение.

Утверждение. Треугольник (невырожденный) можно построить тогда и только тогда, когда все стороны положительны и удовлетворяют неравенству треугольника – сумма каждой двух сторон должна быть больше третьей.

Доказательство. (Необходимость) Пусть треугольник со сторонами a, b, c может быть построен. Построим его и обозначим его вершины A, B, C . Докажем, например, что $a + b > c$. Так как треугольник невырожденный, то C не лежит на прямой AB . Опустим перпендикуляр CD на эту прямую.

Если оба угла при основании треугольника ($\angle BAC$ и $\angle ABC$) острые (или один из них прямой), то точка D будет принадлежать отрезку AB и, поэтому $AB = AD + DB$. В случае, когда один из углов тупой, точка D попадет на продолжение отрезка AB и поэтому $AB < AD + DB$. В любом случае $AB \leq AD + BD$. Поскольку наклонная всегда больше своей проекции, имеем $AD < AC, BD < BC$. Отсюда следует, что $c = AB < AC + BC = a + b$. Аналогично доказываются неравенства и для других пар сторон.

(Достаточность) Пусть заданы три положительные стороны a, b, c , удовлетворяющие неравенству треугольника. Докажем, что существует треугольник с заданными сторонами. Доказательство будет конструктивным, то есть мы укажем способ построения такого треугольника.

Откладываем произвольным образом отрезок $AB = c$. Из точки A проведем окружность радиуса b , а из точки B – окружность радиуса a .

Обозначим A_1, A_2 — точки пересечения первой окружности с прямой AB , а B_1, B_2 — второй окружности. Тогда $AB_1 = |AB - BB_1| = |a - c|$ (знак модуля взят, потому что при $a > c$ выражение может оказаться отрицательным, в этом случае точка B_1 окажется левее A). Из неравенства треугольника $c < a + b$ и $a < b + c$ следует, что $c - a < b$ и $a - c < b$, а значит $AB_1 = |a - c| < b$, то есть расстояние от B_1 до центра первой окружности меньше ее радиуса, и, следовательно, точка лежит внутри круга. С другой стороны, $AB_2 = AB + BB_2 = c + a > b$, и значит точка B_2 лежит вне круга. Поскольку окружность — это непрерывная кривая, вторая окружность должна пересечь первую, а в силу замкнутости точек пересечения будет 2, и они будут лежать в разных полуплоскостях относительно прямой AB . Обозначим точки пересечения этих окружностей C и C' . Нетрудно видеть, что $\triangle ABC$ (равно как и $\triangle ABC'$) имеет стороны равные a, b, c . Действительно, $AB = c$ по построению, $AC = b$ как радиус первой окружности, $BC = a$ как радиус второй окружности. Тем самым утверждение доказано.

Заметим, что при фиксированном выборе местоположения отрезка $AB = c$, и определения, от какого из его концов какой отрезок (a или b) следует откладывать, а также выбора полуплоскости, в которой будет располагаться третья точка, треугольник строится однозначно.

Заметим, что проверка положительности сторон является излишней. Действительно, пусть выполняются неравенства $a + b > c$ и $a + c > b$. Тогда мы можем их сложить, получим $2a + b + c > c + b$. Затем вычтем из обоих частей $b + c$ и разделим на положительное число 2. Получаем, что $a > 0$. Аналогично можно показать, что $b > 0$ и $c > 0$. Это позволяет нам избавиться от трех операций сравнения.

Все три неравенства для треугольника эквивалентны одному неравенству в следующей форме:

$$a + b + c > 2 \max(a, b, c).$$

Для треугольника такой вид не дает никаких преимуществ, поскольку еще несколько проверок потратится для нахождения максимума, однако аналогичное неравенство в задаче о многоугольнике оказывается достаточно удобным.

Перейдем теперь непосредственно к построению треугольника. Будем искать его в виде $A_1 = (x_1, 0), A_2 = (x_2, 0), A_3 = (0, y)$, где $x_1 < x_2, y > 0$. Такой треугольник находится однозначно из уравнений:

$$x_2 - x_1 = a, \quad x_2^2 + y^2 = b^2, \quad x_1^2 + y^2 = c^2.$$

Откуда находим

$$x_2 + x_1 = d = (b^2 - c^2)/a,$$

и, следовательно,

$$x_1 = (d - a)/2, \quad x_2 = (d + a)/2, \quad y = \sqrt{c^2 - x_1^2}.$$

Из неравенства треугольника следует, что величина d по модулю не превосходит $b + c$.

Рассмотрим теперь задачу в варианте High о существовании многоугольника.

Утверждение. N -угольник можно построить тогда и только тогда, когда все стороны положительны и каждая сторона меньше суммы остальных $N - 1$ сторон.

Доказательство. Воспользуемся методом математической индукции. Для $N = 3$ утверждение уже доказано. Пусть утверждение выполнено для всех $N < k$, докажем, что из этого следует выполнение его для $N = k$.

(Необходимость.) Пусть построен k -угольник со сторонами a_1, a_2, \dots, a_k . Соединим вершины A_{k-1} и A_1 , диагональю x . По предположению индукции для сторон $(k - 1)$ -угольника $A_1 A_2 \dots A_{k-1}$ выполняется неравенство $x < a_1 + a_2 + \dots + a_{k-2}$. А из треугольника $A_1 A_{k-1} A_k$ получаем $a_k < x + a_{k-1}$ или (с учетом предыдущего неравенства) $a_k < a_1 + a_2 + \dots + a_{k-1}$. Аналогично доказывается неравенство и для остальных сторон.

(Достаточность.) Пусть у нас есть k сторон a_1, a_2, \dots, a_k . Для определенности будем считать, что a_k — наибольшая сторона. Тогда построим треугольник со сторонами a_{k-1}, a_k, x , и $(k - 1)$ -угольник со сторонами $a_1, a_2, \dots, a_{k-2}, x$, где x — некоторое вещественное число, которое мы дальше определим так, чтобы соответствующие построения были возможны. Если совместить стороны x полученного треугольника и $(k - 1)$ -угольника так, чтобы они оказались в разных полуплоскостях относительно прямой, проходящей через сторону x , то получим k -угольник с нужными сторонами (однако возможно не являющийся выпуклым). Теперь укажем как следует выбирать значение x . Поскольку все стороны a_i положительны, то, для того чтобы существовал треугольник, необходимо и достаточно, чтобы x удовлетворяло неравенству

$$a_k - a_{k-1} < x < a_k + a_{k-1}. \tag{*}$$

Для того же, чтобы можно было построить $(k - 1)$ -угольник, по предположению индукции необходимо и достаточно, чтобы

$$\begin{aligned} x &> 0, \\ a_1 + a_2 + \dots + a_{k-2} &> x, \\ x + (a_1 + a_2 + \dots + a_{k-2} - l) &> l, \end{aligned}$$

где l — наибольшая среди сторон a_1, a_2, \dots, a_{k-2} . Эти неравенства относительно x могут быть записаны в виде

$$\max\{0, 2l - (a_1 + a_2 + \dots + a_{k-2})\} < x < a_1 + a_2 + \dots + a_{k-2}. \tag{**}$$

В неравенстве (***) левая граница меньше l , а правая — больше. Поэтому множество его решений непусто. В силу условия утверждения для k -угольника $a_1 + a_2 + \dots + a_{k-2} > a_k - a_{k-1}$. Поэтому правая граница неравенства (***)) больше левой границы неравенства (*), и значит общее решение неравенств (*) и (***)) существует и может быть записано как

$$\max\{2l - (a_1 + a_2 + \dots + a_{k-2}), a_k - a_{k-1}\} < x < \min\{a_1 + a_2 + \dots + a_{k-2}, a_k + a_{k-1}\}.$$

Остается показать, что среди значений x , удовлетворяющих неравенству, найдется такое, что результирующий многоугольник будет выпуклым. Если взять x равным $d = \min\{a_1 + a_2 + \dots + a_{k-2}, a_k + a_{k-1}\}$, то треугольник или $(k-1)$ -угольник окажется вырожденным, но в итоге получим, что углы при стыковке строго меньше 180 градусов. При любом $\varepsilon > 0$ для $x = d - \varepsilon$ уже будем иметь невырожденные треугольник и $(k-1)$ -угольник, но при достаточно малом ε углы стыковки по прежнему останутся меньше, чем 180 градусов. Значит он будет выпуклым и невырожденным, что и требовалось доказать.

Следует отметить, что для многоугольника с числом сторон больше трех, условие положительности сторон не следует из неравенства, накладывающего ограничения на сумму сторон. Например, сумма любых трех чисел из $-1, 3, 4, 5$ больше четвертого, однако среди них есть отрицательное число -1 . Поэтому в алгоритме следует искать не только максимальное значение, но и минимальное, проверяя его знак.

Несмотря на то, что доказательство возможности построения было конструктивным, использование этого способа для построения упирается в проблему выбора длины диагонали. Выбор достаточно близкого к d значения приводит к близким развернутым углам, и к быстрому увеличению диаметра многоугольника.

Чтобы избежать этой проблемы, будем искать многоугольник с заданными сторонами, который будет вписан в некоторую окружность. Найдем диаметр d такой окружности. Если ее центр лежит внутри многоугольника (или на границе), то сумма углов, под которыми будут видны из центра стороны многоугольника, должна быть равна 2π :

$$f(d) = 2 \sum_{1 \leq i \leq N} \arcsin \frac{a_i}{d} = 2\pi.$$

Минимальное допустимое значение d , при котором определена функция f , равно $\max a_i$ (очевидно в окружность с меньшим диаметром вписать многоугольник не получится — не войдет большая сторона). Функция f — убывающая, и ее значение при $d \rightarrow +\infty$ стремится к нулю. Поэтому если $f(\max a_i) > 2\pi$, то найдется значение d , при котором достигается равенство.

Такое значение d может быть найдено с помощью бинарного поиска. Вершины искомого многоугольника будут тогда иметь координаты:

$$x_i = d/2 \cdot \cos \varphi_i, y_i = d/2 \cdot \sin \varphi_i,$$

где $\varphi_1 = 0, \varphi_i = \varphi_{i-1} + 2 \arcsin \frac{a_{i-1}}{d}$.

Если же $f(\max a_i) < 2\pi$, то искомая окружность будет иметь центр вне многоугольника, в этом случае угол, под которым видна максимальная сторона (легко показать, что такая будет только одна) должен быть равен сумме углов, под которыми видны остальные стороны:

$$f_2(d) = 2 \left(\sum_{1 \leq i \leq N} \arcsin \frac{a_i}{d} - 2 \arcsin \frac{\max a_i}{d} \right).$$

Значение $f_2(\max a_i)$ в таком случае будет отрицательным, при $d \rightarrow +\infty$ имеет место равенство $f_2(d) = d^{-1}(\sum a_i - 2 \max a_i) + o(d^{-2})$. Следовательно, при достаточно больших d функция f_2 принимает положительные значения, а, значит, существует корень уравнения $f_2(d) = 0$, для нахождения которого может быть применен какой-нибудь численный метод (например, опять же бинарный поиск).

Без ограничения общности, можно считать, что максимальная сторона – это сторона с номером N . В этом случае формула для координат вершин многоугольника останется справедливой.

Общая сложность вычислений составит порядка $O(N \log \frac{Nm}{\varepsilon})$, где ε – требуемая точность, m – максимальное значение длин сторон.

Задача Е. Красивые узоры (высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Компания BrokenTiles планирует заняться выкладыванием в некотором дворе узора из черных и белых плиток, каждая из которых имеет размер 1×1 метр. Двор имеет форму прямоугольника $N \times M$ метров. Однако оказалось, что K плиток уже расположены в некоторых клетках двора. Требуется определить, сколько вариантов узора может быть выполнено на этом дворе, при условии, что узор должен быть красивым.

Узор считается красивым, если в любом квадрате 2×2 есть либо три черных плитки и одна белая или, наоборот, одна черных и три белых.

Ограничения

$1 \leq N, M \leq 10^5$, $0 \leq K \leq 10^5$.

Формат входного файла

В первой строке даны три целых числа N , M и K . В каждой из последующих K строк записаны по три целых числа x , y , c ($1 \leq x \leq N$, $1 \leq y \leq M$), обозначающих, что в клетке с координатами (x,y) находится плитка. Если она черная, то $c = 0$, а если белая, то $c = 1$. Все клетки — различные.

Формат выходного файла

Выведите одно число — остаток от деления на $10^9 + 7$ количества различных красивых узоров, которые могут быть выполнены на данном дворе. Узоры считаются различными, если есть хотя бы одна клетка, в которой в одном узоре лежит белая плитка, а в другом — черная.

Пример

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

Задача F. Красивые узоры (юниорская лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Компания BrokenTiles планирует заняться выкладыванием в некотором дворе узора из черных и белых плиток, каждая из которых имеет размер 1×1 метр. Двор имеет форму прямоугольника $N \times M$ метров. Однако оказалось, что K плиток уже расположены в некоторых клетках двора. Требуется определить, сколько вариантов узора может быть выполнено на этом дворе, при условии, что узор должен быть красивым.

Узор считается красивым, если в любом квадрате 2×2 есть либо три черных плитки и одна белая или, наоборот, одна черных и три белых.

Ограничения

$$1 \leq N, M \leq 10, 0 \leq K \leq NM.$$

Формат входного файла

В первой строке даны три целых числа N, M и K . В каждой из последующих K строк записаны по три целых числа x, y, c ($1 \leq x \leq N, 1 \leq y \leq M$), обозначающих, что в клетке с координатами (x, y) находится плитка. Если она черная, то $c = 0$, а если белая, то $c = 1$. Все клетки — различные.

Формат выходного файла

Выведите одно число — количество различных красивых узоров, которые могут быть выполнены на данном дворе. Узоры считаются различными, если есть хотя бы одна клетка, в которой в одном узоре лежит белая плитка, а в другом — черная.

Пример

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

Разбор задачи F. Красивые узоры

Рассмотрим сначала случай, когда ни одна клетка не окрашена изначально. Обозначим через A_{xy} — цвет клетки с координатами (x, y) . Очевидно условие на красоту определяется равенством:

$$A_{ij} + A_{i+1,j} + A_{i,j+1} + A_{i+1,j+1} = 1,$$

где сложение понимается по модулю 2.

Таким образом, у нас имеется $(N - 1)(M - 1)$ уравнений с NM параметрами. Нетрудно проверить, что система этих уравнений совместна. Действительно, значения $A_{ij} = ij \bmod 2$ удовлетворяет всем уравнениям. Следовательно, красивые раскраски существуют. Кроме того, поскольку система недоопределенна, должно быть некоторое количество d переменных, которые можно выбрать произвольно, а остальные однозначно определяются из них. При этом $d \geq NM - (N - 1)(M - 1) = N + M - 1$. С другой стороны,

если мы зададим $N + M - 1$ значений A_{x1} и A_{1y} , то все остальные переменные смогут быть однозначно вычислены. Таким образом, $d = N + M - 1$. Это означает, что все исходные уравнения являются независимыми, и общее количество красивых узоров для таблицы заданных размеров равно 2^{N+M+1} .

Рассмотрим теперь случай, когда ряд плиток уже имеют некоторый цвет. Подставим их значения в соответствующие уравнения. Количество переменных в этом случае уменьшится на k . Система теперь может оказаться переопределённой, а может по-прежнему иметь некоторое количество независимых переменных. Для их определения воспользуемся методом Гаусса, приведя матрицу коэффициентов системы к диагональному виду. Если на каком-то шаге получится уравнение с нулевыми коэффициентами, а в его правой части будет 1, это будет обозначать, что система несовместна, и нет ни одного красивого узора для такой заданной начальной раскраски. В противном случае, у нас будет некоторое количество d независимых переменных, которые могут выбираться абсолютно произвольно. Ответом на задачу будет -2^d .

Временная сложность такого алгоритма $(NM)^3$, а пространственная $(NM)^2$. Это приемлемо для варианта задачи Junior. Впрочем, здесь вполне допустимы и варианты решения динамическим программированием по профилю (сложность $O(2^{2N}M)$), с ломанным профилем ($O(2^N NM)$) и даже обычный перебор вариантов заполнения первой строки и столбца (сложность $O(2^{N+M-1} NM)$).

Для решения варианта High заметим, что каждая окрашенная клетка может уменьшить количество независимых переменных (изначально равное $d = N + M - 1$) на 1. За исключением случая, когда значение этой переменной уже определилось из предыдущих уравнений системы. Это возможно лишь в том случае, когда клетка является вершиной некоторой фигуры, состоящей из ребер, параллельных осям координат и имеющей только прямые углы. Легко проверить, что любая такая фигура замощается квадратами 2×2 (их количество равно площади соответствующей фигуры (если считать вершинами центры клеток), при этом каждая клетка будет покрыта четным количеством таких квадратов, за исключением этих самых угловых клеток. Отсюда следует, что сумма значений в угловых клетках такой фигуры должна быть сравнима по модулю 2 с площадью этой фигуры. Так что если некоторая клетка “закрывает” одну из таких фигур, то нужно проверить соответствующее равенство, а количество независимых переменных останется прежним. Для обнаружения таких клеток удобно представить информацию об окраске в виде двудольного графа. Вершины одной его доли будут соответствовать вертикалям, а другой — горизонтальм, окрашенным клеткам будут соответствовать ребра, соединяющие соответствующие вершины. Тогда каждый цикл в таком графе будет соответствовать фигуре в таблице. Для поиска циклов, как известно, можно применить поиск в глубину. Если при таком поиске находится обратное реб-

ро, то его (после проверки совместности) удаляем из графа.

Общая сложность алгоритма оценивается величиной $O(N + M + k)$.

Задача G. Стока без повторений (высшая лига)

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Рассмотрим строку символов. Будем говорить, что в строке $s_1 s_2 \dots s_n$ есть повторение, если в ней есть две совпадающие подстроки, следующие непосредственно одна за другой. То есть, если для некоторых i и k ($i, k > 0$, $i + 2k - 1 \leq n$) выполняется $s_i = s_{i+k}, s_{i+1} = s_{i+k+1}, \dots, s_{i+k-1} = s_{i+2k-1}$.

Найдите строку длины n без повторений с минимальным количеством использованных букв.

Ограничения

$$1 \leq n \leq 4 \cdot 10^6.$$

Формат входного файла

В единственной строке задается одно целое число n — длина искомой строки.

Формат выходного файла

В первой строке выведите минимальное количество различных символов, которые нужно использовать для построения строки без повторений, а во второй строке — искомую строку. Разрешается использовать лишь маленькие латинские буквы. Гарантируется, что при всех входных данных 26 символов будет достаточно для построения строки без повторений (возможно, что не оптимальной).

Пример

stdin	stdout
5	3 abaca

Задача Н. Стока без повторений (юниорская лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Рассмотрим строку символов. Будем говорить, что в строке $s_1 s_2 \dots s_n$ есть повторение, если в ней есть две совпадающие подстроки, следующие непосредственно одна за другой. То есть, если для некоторых i и k ($i, k > 0$, $i + 2k - 1 \leq n$) выполняется $s_i = s_{i+k}$, $s_{i+1} = s_{i+k+1}$, \dots , $s_{i+k-1} = s_{i+2k-1}$.

Найдите лексикографически минимальную строку длины n без повторений.

Ограничения

$1 \leq n \leq 4 \cdot 10^6$.

Формат входного файла

В единственной строке задается одно целое число n — длина искомой строки.

Формат выходного файла

Выведите одну искомую строку. Разрешается использовать лишь маленькие латинские буквы. Гарантируется, что при всех входных данных 26 символов будет достаточно для построения строки без повторений (возможно, что не оптимальной).

Пример

stdin	stdout
5	abaca

Разбор задачи Н. Стока без повторений

Для решения варианта Junior заметим, что любой префикс p оптимальной строки s_n без повторений размера n является оптимальной строкой без повторений меньшего размера (обозначим его m). Поскольку во всей строке s нет повторений, нет их и в p . Очевидно, оптимальная строка s_m размера m тогда не может быть лексикографически больше p . Предположим, что s_m меньше p , тогда допишем к ней последовательно символы с кодами $c+1, c+2, \dots$, до размера n , где c — код максимального символа в этой строке. В полученной строке по-прежнему не будет повторений, и она будет лексикографически меньше s_n , что противоречит оптимальности s_n .

Отсюда следует, что достаточно построить один раз оптимальную строку достаточно большого размера, а затем выдавать ее префикс нужного размера.

Можно показать по индукции, что это будет строка вида “abacabadabacaba....”, в которой на нечетных позициях (то есть с номерами вида $2k + 1$) стоит символ a , на местах с номерами вида $4k + 2$ стоит символ b и т.д. На позициях с номерами $2^l k + 2^{l-1}$ будет стоять l -ый символ алфавита.

Для варианта High всегда можно использовать не более трех символов. Причем, непосредственно проверяется, что при $N = 1$ достаточно одного символа, а при $N = 2$ или $N = 3$ – двух.

Как и в варианте Junior, опять же можно построить одну большую строку без повторений и выводить ее префиксы. Существует несколько способов построения требуемой такой строки, приведем один из них. Здесь будут указаны основные утверждения, которые используются для построения и проверки отсутствия повторений.

Последовательностью Морса-Туэ – это последовательность $\{s_i\}$, $i = 0, 1, \dots$, состоящая из двух символов a и b , которая определяется следующим образом: $s_i = a$, если i содержит четное количество единиц в двоичном представлении, и $s_i = b$, если i содержит нечетное количество единиц в двоичном представлении.

Начало этой последовательности – “abbabaabbaabab...”. Для удобства введем одноместную операцию “–” так, что $-a = b$, $-b = a$.

Запишем некоторые свойства этой последовательности:

Свойство 1. $s_{2i} = s_i$.

Свойство 2. $s_{2i+1} = -s_i$.

Свойства следуют непосредственно из определения.

Свойство 3. Если $s_j = s_{j+1}$, то j – нечетное.

Однако обратное утверждение в общем случае неверно: не для всякого нечетного j выполнено $s_j = s_{j+1}$ (например, при $j = 3$).

Доказательство. Предположим j – четное. Но тогда по свойству 1 $s_j = s_{j/2}$, а по свойству 2 $s_{j+1} = -s_{j/2}$, откуда следует, что $s_j \neq s_{j+1}$, что противоречит условию.

Свойство 4. Не существует такого j , что $s_j = s_{j+1} = s_{j+2}$.

Это свойство непосредственно следует из свойства 3, поскольку хотя бы одно из чисел j или $j + 1$ обязано быть нечетным.

Для заданной строки t обозначим строку из символов, стоящих на четных местах через $p(t)$, а на нечетных – $n(t)$. Нетрудно видеть, что $p(s) = s = -n(s)$.

Свойство 5. Среди любых пяти последовательных символов строки s найдется по крайней мере одна пара равных последовательных символов.

Доказательство. Существует всего две последовательности из пяти символов, в которых нет подряд стоящих одинаковых символов “ababa” и “babab”. Рассмотрим подпоследовательность состоящую из трех символов с номерами 1, 3, 5. Она будет тогда равна либо “aaa”, либо “bbb”, и являться подстрокой либо $p(s)$, либо $n(s)$. Но по свойству 4 ни одна из этих ситуаций не является возможной.

Утверждение 1. В последовательности Морса-Туэ нет повторений с накладкой на последний символ, то есть не существует таких $i \geq 0$ и $l > 0$, что $s[i : i + l] = s[i + l : i + 2l]$.

Доказательство. По свойству 4 значение l не может быть равно 1. Рассмотрим случай, когда l нечетное, большее 1. Тогда в подстроке $s[i : i + 2l]$ есть не меньше 7 букв, и значит по свойству 5, найдется такое $i \leq j < i + 2l$, что $s_j = s_{j+1}$. Значение j не может быть четным по свойству 3. Значит j нечетно. Обе эти буквы входят в одну из подстрок $s[i : i + l]$ или $s[i + l : i + 2l]$. Но так как эти строки равны, то есть такая же пара символов и во второй строке на позиции $j + l$ или $j - l$. Но оба этих числа будут четными, что опять же противоречит свойству 3.

Перейдем к случаю, когда l четно. Символы с номерам $i, i + 2, \dots, i + l, i + l + 2, \dots, i + 2l$ попадают либо в строку $p(s)$ либо в $n(s)$, для которой будет выполняться аналогичное свойство с $l/2$. Если это значение опять же окажется четным, то можно будет снова применить такую же процедуру, уменьшив значение до $l/4$. Но когда-нибудь это число станет нечетным. А по доказанному выше, не существует повторений с накладкой нечетной длины. Утверждение доказано.

Рассмотрим новый алфавит $\Sigma = \{[aa], [ab], [ba], [bb]\}$ и построим в нем последовательность $\{d_i\}$, где $d_i = [s_i s_{i+1}]$ для каждого $i \geq 0$.

Утверждение 2. Эта последовательность является последовательность без повторений.

Нетрудно проверить, что любая пара последовательных строк в d соответствует паре с накладкой в строке s .

Итак мы показали, что существует последовательность без повторений из четырех различных символов. Для того, чтобы построить трехбуквенную последовательность достаточно заметить, что по свойству 3 значение [aa] всегда идет следом за [ba], а предшествует ему обязательно [ab]. Кроме того, [bb] всегда идет следом за [ab], но перед [ba]. Следовательно, отождествив символы [aa] и [bb] мы оставим ту же строку по прежнему без повторений. Это и будет искомая последовательность без повторений, в которой используются лишь 3 различных символа.

Задача I. Сбор бобов (высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Есть бесконечная в одну сторону полоса, состоящая из клеток, пронумерованных числами 0, 1, 2, В некоторых клетках лежит некоторое количество бобов. Каждый ход разрешается выполнять следующее действие. Если в клетке с номером i ($i > 0$) есть не менее i горошин, то i из них забираются из этой клетки и раскладываются по одной в клетки с номерами $i - 1, i - 2, \dots, 0$. Если есть несколько таких клеток, то можно делать ход из любой из них. Ваша задача — собрать в клетке с номером 0 все имеющиеся на полосе бобы.

Ограничения

$$0 \leq N \leq 2 \cdot 10^5, 0 \leq i_k \leq 10^9, 1 \leq a_k \leq 10^{18}.$$

Формат входного файла

В первой строке задано одно целое число N . В каждой из последующих N строк задаются по два целых числа i_k и a_k , означающих, что в клетке с номером i_k находится в начальном состоянии a_k горошин. Гарантируется, что все i_k различны.

Формат выходного файла

Выведите “Yes”, если существует такая последовательность действий, которая приведет к тому, что все горошины окажутся в клетке 0. В противном случае нужно вывести “No”.

Пример

stdin	stdout
2 1 1 2 2	Yes
3 1 1 2 2 3 3	No
4 0 3 1 3 2 3 3 3	Yes

Задача J. Сбор бобов (юниорская лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Есть полоса, состоящая из $N + 1$ клеток, пронумерованных числами 0, 1, 2, ..., N . В каждой из клеток лежит некоторое количество бобов. Каждый ход разрешается выполнять следующее действие. Если в клетке с номером i ($i > 0$) есть не менее i горошин, то i из них забираются из этой клетки и раскладываются по одной в клетки с номерами $i - 1, i - 2, \dots, 0$. Если есть несколько таких клеток, то можно делать ход из любой из них. Ваша задача — собрать в клетке с номером 0 все имеющиеся на полосе бобы.

Ограничения

$$0 \leq N \leq 10^5, 0 \leq a_i \leq 10^9.$$

Формат входного файла

В первой строке задано одно целое число N . Во второй строке записаны $N + 1$ целых чисел a_0, a_1, \dots, a_N , где a_i — количество горошин в клетке с номером i .

Формат выходного файла

Выведите “Yes”, если существует такая последовательность действий, которая приведет к тому, что все горошины окажутся в клетке 0. В противном

случае нужно вывести “No”.

Пример

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

Разбор задачи J. Сбор бобов

Очевидно, следует выполнять описанную операцию, начиная с непустых клеток с максимальными номером. Действительно, в этой клетке у нас никогда не появится новых горошин. Поэтому, если мы не сможем ее “разгрузить” сразу, то этого не удастся сделать никогда. Таким образом, если a_N не кратно N , то ответ будет сразу “No”. Если же a_N кратно N , то во все клетки с номером от 0 до $N - 1$ добавится по a_N/N горошин. Выполнив это, мы получим состояние, в котором непустой клеткой с максимальным номером будет $N - 1$. Продолжая действовать так дальше, мы либо на очередном шаге получим значение a_i не кратное i , и тогда остановимся с ответом “No”, либо дойдем до клетки с номером 0, что будет обозначать ответ “Yes” поскольку мы добились поставленной цели.

Прямая реализация этого алгоритма потребует порядка $O(N^2)$ операций, что очень много даже для варианта Junior. Чтобы уменьшить вычислительную сложность, следует использовать отложенное добавление бобов, пришедших из ячеек с большими номерами. Выручит то, что в каждый момент времени эта добавка является одинаковой для всех ячеек с номерами меньшими, чем текущая рассматриваемая ячейка. Это дает алгоритм со сложностью $O(N)$.

Для варианта High эта оценка превращается в $O(\max i_k)$, что кажется весьма большой величиной. Однако в действительности за счет того что количество непустых ячеек ограничено, мы либо быстро пройдем их все, либо наткнемся на достаточно большой промежуток, в котором нет ни одной горошины. Для того, чтобы “пройти” этот промежуток без потери делимости, потребуется, чтобы в текущей непустой ячейке i было количество горошин кратное $i \cdot (i - 1) \cdot \dots \cdot (i - k)/2^{k/2}$, где k – длина промежутка из пустых ячеек. Поскольку общее количество бобов на полосе ограничено значением 10^{23} , эта длина не может превышать 27 даже для не слишком больших номеров i . Так что идея решения сохраняется. Единственное о чем следует позаботиться – о возможности работы с числами имеющими порядок 10^{23} .

Задача К. Обратный сбор бобов (высшая лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Есть бесконечная в одну сторону полоса, состоящая из клеток, пронумерованных числами 0, 1, 2, У вас есть K горошин, которые вы должны разместить в клетках этой полосы. Необходимо сделать это таким образом, чтобы существовала последовательность действий, которая приведет к тому, что все K бобов окажутся в клетке с номером 0. Напомним, что разрешенным является следующее действие. Если в клетке с номером i ($i > 0$) есть не менее i горошин, то i из них забираются из этой клетки и раскладываются по одной в клетки с номерами $i - 1, i - 2, \dots, 0$. Если есть несколько таких клеток, то можно делать ход из любой из них.

Ограничения

$$0 \leq K \leq 10^{11}.$$

Формат входного файла

В единственной строке записано одно целое число K .

Формат выходного файла

Выведите N строк, где N – количество ячеек, в которых будет ненулевое количество горошин. В каждой строке должны быть указаны два целых числа i_k и a_k , означающих, что в клетке с номером i_k находится в начальном состоянии a_k горошин. Строки должны выводиться в порядке увеличения i_k .

Если существует несколько вариантов расположения, выберите антилексикографически минимальный. Напомним, что набор $(x_0, x_1, \dots, x_i, \dots)$ антилексикографически предшествует $(y_0, y_1, \dots, y_i, \dots)$, если существует такое i , что $x_i > y_i$ и $x_j = y_j$ для всех $j > i$.

Пример

stdin	stdout
3	1 1 2 2
7	1 1 3 2 4 4

Задача L. Обратный сбор бобов (юниорская лига)

Вход:	stdin
Выход:	stdout
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Есть бесконечная в одну сторону полоса, состоящая из клеток, пронумерованных числами 0, 1, 2, У вас есть K горошин, которые вы должны разместить в клетках этой полосы. Необходимо сделать это таким образом, чтобы существовала последовательность действий, которая приведет к тому, что все K бобов окажутся в клетке с номером 0. Напомним, что разрешенным является следующее действие. Если в клетке с номером i ($i > 0$) есть не менее i горошин, то i из них забираются из этой клетки и раскладываются по одной в клетки с номерами $i - 1, i - 2, \dots, 0$. Если есть несколько таких клеток, то можно делать ход из любой из них.

Ограничения

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

Формат входного файла

В единственной строке записано одно целое число K .

Формат выходного файла

Выведите N строк, где N — количество ячеек, в которых будет ненулевое количество горошин. В каждой строке должны быть указаны два целых числа i_k и a_k , означающих, что в клетке с номером i_k находится в начальном состоянии a_k горошин. Строки должны выводиться в порядке увеличения i_k .

Если существует несколько вариантов расположения, выберите антилексикографически минимальный. Напомним, что набор $(x_0, x_1, \dots, x_i, \dots)$ антилексикографически предшествует $(y_0, y_1, \dots, y_i, \dots)$, если существует такое, что i , что $x_i > y_i$ и $x_j = y_j$ для всех $j > i$.

Пример

stdin	stdout
3	1 1 2 2
7	1 1 3 2 4 4

Разбор задачи L. Обратный сбор бобов

Прежде всего следует заметить, что в оптимальной расстановке не может быть клеток, в которых количество бобов превышает их номера. Действительно, если в клетке с номером i есть по крайней мере $i + 1$ бобов, то уменьшим там их количество на $i + 1$, а увеличим количество в клетке с номером $i + 1$. Эта расстановка будет антилексикографически меньше. При этом, если любая последовательность действий по отношению к первой расстановке, в которой есть действие с i -ой ячейкой, будет допустима и для второй расстановки, с заменой самого первого действия над ячейкой i на действие над ячейкой $i + 1$, причем приведут эти действия к одному и тому же результату.

Далее по индукции показывается, что такая расстановка является единственной. При этом, чтобы получить из расстановки для k бобов расстановку для $k + 1$ бобов, нужно добавить 1 в ячейку с номером 0, но это нарушит условие $a_0 \leq 0$. Значит нужно (по описанному выше принципу) забрать отсюда одну горошину и перенести ее в клетку 1. Если после этого нарушится свойство $a_1 \leq 1$, то нужно перенести 2 горошины отсюда в клетку 2 и т.д. Когда-нибудь мы выполним перенос i бобов в клетку с номером i , которая в предыдущей расстановке была пустой. Поэтому мы получим корректную расстановку с указанным свойством для $k + 1$, которая по доказанному является единственной.

Эта идея позволяет записать решение со сложностью $O(kN)$, где N — максимальный номер ячейки в корректной расстановке для k бобов. Как показывают расчеты величина N имеет порядок $O(\sqrt{k})$. Однако указанная оценка довольно пессимистична. В действительности, можно показать, что элемент с номером 1 будет меняться на всех k шагах, элемент с номером 2 примерно на $k/2$ шагах, элемент с номером 3 примерно на $k/3$ итерациях. И значит нашу оценку можно уточнить до $O(k \log k)$. Так что для варианта Junior это решение является достаточно быстрым.

Для варианта High построим линейный по количеству задействованных ячеек алгоритм. Заметим, что при каждой операции число бобов на клетках с номерами i и больше либо не меняется, либо уменьшается на i . Поэтому исходное их количество делится на i . Рассмотрим ячейку с номером 1, в ней может находиться либо 0 или 1 горошина. Из условия, что $k - a[1]$ должно делиться на 2, это количество определяется однозначно: $a[1] = k \bmod 2$. После вычитания из k значения $a[1]$ сможем определить значение $a[2] = k \bmod 3$ и т.д. Общая сложность алгоритма будет оцениваться как $O(N) = O(\sqrt{k})$.

Задача M. Уравнение (высшая лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Имеется уравнение

$$a_N x^N + a_{N-1} x^{N-1} + \dots + a_1 x + a_0 = y p^2,$$

в котором a_i , N и p — известные целые числа, а x и y — неизвестные целочисленные переменные, причем $x \geq 0$. Требуется проверить существует ли у данного уравнения решение в целых числах.

Ограничения

$0 \leq N \leq 20$, $|a_i| \leq 10^9$, $2 \leq p \leq 10^6$, p — простое число.

Формат входного файла

В первой строке записаны два целых числа p и N . Вторая строка содержит $N + 1$ чисел a_N, \dots, a_0 .

Формат выходного файла

Если уравнение не имеет решений в целых числах, выведите число -1 . В противном случае выведите значение x из пары (x, y) , удовлетворяющей уравнению. Если таких решений несколько, выберите такое, в котором x принимает наименьшее неотрицательное значение.

Пример

stdin	stdout
3 2	3
1 2 3	
2 2	-1
1 1 1	

Задача N. Уравнение (юниорская лига)

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

Имеется уравнение

$$a_N x^N + a_{N-1} x^{N-1} + \dots + a_1 x + a_0 = y p^2,$$

в котором a_i , N и p — известные целые числа, а x и y — неизвестные целочисленные переменные, причем $x \geq 0$. Требуется проверить существует ли у данного уравнения решение в целых числах.

Ограничения

$$0 \leq N \leq 20, |a_i| \leq 10^9, 1 \leq p \leq 2014.$$

Формат входного файла

В первой строке записаны два целых числа p и N . Вторая строка содержит $N + 1$ чисел a_N, \dots, a_0 .

Формат выходного файла

Если уравнение не имеет решений в целых числах, выведите число -1 . В противном случае выведите значение x из пары (x, y) , удовлетворяющей уравнению. Если таких решений несколько, выберите такое, в котором x принимает наименьшее неотрицательное значение.

Пример

stdin	stdout
3 2	3
1 2 3	
2 2	-1
1 1 1	

Разбор задачи N. Уравнение

Нетрудно убедиться в том, что, если некоторое значение $x \geq p^2$ удовлетворяет уравнению при некотором y , то ему также будет удовлетворять и значение $x - p^2$ (разумеется при другом y). Поэтому оптимальное решение следует искать лишь до значения $p^2 - 1$. Если до этого момента не обнаружится решений,

то их и нет вовсе. В связи с вышесказанным все вычисления следует производить по модулю p^2 . В результате получим алгоритм со сложностью $O(Np^2)$. Это решение вполне допустимо для варианта Junior.

В варианте High представим искомое значение в виде $x = x_1p + x_2$. Подставим это значение в полином, стоящий в левой части уравнения. Получим

$$f(x_1p + x_2) = (\sum ia_i x_2^{i-1})x_1p + \sum a_i * x_2^i.$$

Здесь отброшены все члены, содержащие p в степени не ниже 2.

Отсюда видно, что значение полинома на x_2 должно быть кратно p . Переberем все возможные значения x_2 в пределах от 0 до $p - 1$, вычисляя значения полинома по модулю p . Если в некоторой точке x_2 оно окажется равным 0, то пересчитаем его еще раз, но уже по модулю p^2 . Если и тогда оно равно нулю, то x_2 является ответом. В противном случае, имеем выражение

$$(\sum ia_i x_2^{i-1})x_1 + (f(x_2) \bmod p^2)/p,$$

которое должно быть кратно p . Вычислим коэффициент при x_1 (фактически это производная от многочлена в точке x_2) по модулю p . Если он окажется равен нулю, то решений с таким x_2 не существует. В противном случае, можно найти однозначно найти x_1 , решив тождество

$$f'(x_2)x_1 \equiv -(f(x_2) \bmod p^2)/p \pmod{p}.$$

Считаем с помощью алгоритма Евклида обратный элемент к $f'(x_2)$ по модулю p и домножаем полученное значение на правую часть. Получившееся для одного значения x_2 решение еще не является ответом к задаче — возможно для другого x_2 величина x_1 может получиться меньшей.

Количество действий, выполняемых в этом решении, можно оценить как $O(Np \log p)$, что конечно же является очень пессимистичной оценкой, поскольку вычислять обратный элемент нам придется лишь для тех значений x_2 , которые обращают в 0 полином по модулю p . Вероятнее всего таких будет не слишком много.

Задача О. Соревнование (высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

На соревновании по карате участвуют две команды A и B . В каждой из них есть по N каратистов с известным уровнем мастерства. Соревнование состоит

из N раундов-поединков, в каждом из которых участвует по одному каратисту от каждой команды. Согласно правилам, каждый каратист может поучаствовать только в одном поединке. Каждый раунд заканчивается победой одного из участников (того, у которого выше уровень мастерства), либо ничьей (если уровни мастерства участников одинаковы). Если зафиксирована ничья в поединке, обе команды за раунд получают по 1 очку, в противном случае победитель приносит своей команде 2 очка, проигравший — 0. В начале соревнования тренеры обеих команд дают организаторам списки, где указывают, в каком порядке будут выходить на ринг их подопечные. Однако тренеру команды B стало известно, в каком порядке будут выходить участники из команды A . Обладая этой информацией, он хочет составить свой список таким образом, чтобы набрать как можно больше очков, и просит вас помочь ему.

Ограничения

$$1 \leq N \leq 2 \cdot 10^5, 1 \leq a_i, b_i \leq 10^9.$$

Формат входного файла

В первой строке задается одно целое число N . Во второй строке задаются N чисел a_i , каждое из которых определяет уровень мастерства соответствующего каратиста команды A . Значения задаются в том порядке, в котором будут выходить на ринг участники этой команды. В третьей строке задаются N чисел b_i , определяющих уровни мастерства каратистов команды B .

Формат выходного файла

Выведите одно число — максимальное количество очков, которое может набрать команда B за счет выбора тренером этой команды оптимального порядка.

Пример

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

Пояснение

Во втором примере в первом раунде тренер B выпустит каратиста с уровнем 1 (который наберет 0 очков в поединке с каратистом уровня 4), а в после-

дующих — каратистов с уровнями 7, 8 и 3 соответственно, которые выиграют свои поединки против 5, 6 и 2 и принесут по два очка.

Задача Р. Соревнование (юниорская лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

На соревновании по карате участвуют две команды A и B . В каждой из них есть по N каратистов с известным уровнем мастерства. Соревнование состоит из N раундов-поединков, в каждом из которых участвует по одному каратисту от каждой команды. Согласно правилам, каждый каратист может поучаствовать только в одном поединке. Каждый раунд заканчивается победой того из участников, у которого выше уровень мастерства. Победитель приносит своей команде 1 очко, проигравший — 0. В начале соревнования тренеры обеих команд дают организаторам списки, где указывают, в каком порядке будут выходить на ринг их подопечные. Однако тренеру команды B стало известно, в каком порядке будут выходить участники из команды A . Обладая этой информацией, он хочет составить свой список таким образом, чтобы набрать как можно больше очков, и просит вас помочь ему.

Ограничения

$$1 \leq N \leq 2 \cdot 10^5, 1 \leq a_i, b_i \leq 2N.$$

Формат входного файла

В первой строке задается одно целое число N . Во второй строке задаются N чисел a_i , каждое из которых определяет уровень мастерства соответствующего каратиста команды A . Значения задаются в том порядке, в котором будут выходить на ринг участники этой команды. В третьей строке задаются N чисел b_i , определяющих уровни мастерства каратистов команды B . Гарантируется, что уровни мастерства у всех участников различны.

Формат выходного файла

Выведите одно число — максимальное количество очков, которое может набрать команда B за счет выбора тренером этой команды оптимального порядка.

Пример

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

Пояснение

Во втором примере в первом раунде тренер B выпустит каратиста с уровнем 1 (который наберет 0 очков в поединке с каратистом уровня 4), а в последующих — каратистов с уровнями 7, 8 и 3 соответственно, которые выиграют свои поединки против 5, 6 и 2 и принесут по одному очку.

Разбор задачи Р. Соревнование

Упорядочим всех каратистов по их силе. В варианте, когда нет равных по силе каратистов, а значит нет ничьих, решать задачу можно жадным алгоритмом. Для каждого из самых слабых каратистов команды A найдем тех (среди еще незадействованных спортсменов команды B), кто их обыграет. Используя технику двух указателей, это может быть выполнено за один проход по обоим массивам.

Задача несколько усложняется, если могут быть ничьи. Однако можно показать, что оптимальное сочетание можно искать среди таких, у которых все ничейные поединки проходят между спортсменами с одинаковым уровнем мастерства. Действительно, если есть два ничейных поединка, в одном из которых сражаются каратисты с уровнем x , а в другом — с уровнем y , то можно поменять каждому каратисту соперника, получив вместо двух ничьих одну победу и одно поражение, получив те же самые два очка.

Определим f_j — максимальное количество спортсменов из команды A , которых могут обыграть первые j (возможно не все) спортсменов команды B . Все эти значения могут быть вычислены за один проход, с помощью того же самого жадного алгоритма для варианта Junior. Кроме того, посчитаем величины g_i , равные максимальному количеству спортсменов с номерами не меньше i из команды A , которых могут обыграть спортсмены команды B . Это можно сделать аналогичным проходом по массивам, но уже с конца. Тогда $2f_N = 2g_1$ — это максимальное количество очков, которые можно набрать без ничьих.

Теперь рассмотрим варианты с ничьими. Их есть смысл устраивать для таких значений, что $f_j = i$ и при этом $a_{i+1} = b_{j+1} > b_j$. Обозначим значение $v = a_{i+1}$. Тогда если для некоторого значения d выполняется $a_{i+d} = b_{j+d} = v$, то мы можем поступить следующим образом: первые j спортсменов B обыгрывают i спортсменов команды A . Затем пары $(i+1, j+1), \dots, (i+d, i+d)$ дают нам d ничьих, и после этого g_{i+d+1} спортсменов команды A с номерами не ниже $i+d+1$ будут обыграны лучшими спортсменами из команды B . В итоге получим $d + 2(i + g_{i+d+1})$ очков. Нужно учесть эти значения для всех допустимых значений d . Несмотря на то, что мы запускаем такой линейный поиск значений d (еще один цикл), в действительности это не приведет к увеличению сложности алгоритма, поскольку каждое значение v будет рассматриваться лишь один раз, а значит при всех таких поисках мы просмотрим каждый элемент не более одного раза.

Таким образом, в обоих вариантах задача решается за время $O(N)$ (без учета сортировки).

Задача Q. Вписанная окружность (высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Задан выпуклый многоугольник. Будем говорить, что некоторая окружность является вписанной в данный многоугольник, если все ее точки содержатся внутри многоугольника или на его границе. Требуется найти вписанную окружность с наибольшим радиусом.

Ограничения

$$3 \leq N \leq 50000, -10^7 \leq x_i, y_i \leq 10^7.$$

Формат входного файла

В первой строке задается одно целое число N . В каждой из последующих N строк содержатся по два вещественных числа x_i, y_i , имеющих не более 6 знаков после десятичной точки и определяющие координаты соответствующей вершины многоугольника. Вершины многоугольника задаются в порядке обхода. Никакие три точки не лежат на одной прямой.

Формат выходного файла

Выведите одно число — радиус максимальной вписанной окружности с точностью не менее 10^{-5} .

Пример

stdin	stdout
3 2.0 0.0 0.0 0.0 1.0 1.0	0.414214
4 -1 0 0 1 2 -1 1 -2	0.707107

Задача R. Вписанная окружность (юниорская лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Задан выпуклый многоугольник. Будем говорить, что некоторая окружность является вписанной в данный многоугольник, если все ее точки содержатся внутри многоугольника или на его границе. Требуется найти вписанную окружность с наибольшим радиусом.

Ограничения

$$3 \leq N \leq 100, -10^6 \leq x_i, y_i \leq 10^6.$$

Формат входного файла

В первой строке задается одно целое число N . В каждой из последующих N строк содержатся по два вещественных числа x_i, y_i , имеющих не более 6 знаков после десятичной точки и определяющие координаты соответствующей вершины многоугольника. Вершины многоугольника задаются в порядке обхода. Никакие три точки не лежат на одной прямой.

Формат выходного файла

Выведите одно число — радиус максимальной вписанной окружности с точностью не менее 10^{-5} .

Пример

stdin	stdout
3 2.0 0.0 0.0 0.0 1.0 1.0	0.414214
4 -1 0 0 1 2 -1 1 -2	0.707107

Разбор задачи R. Вписанная окружность

Запишем уравнения всех сторон многоугольника в виде

$$a_i x + b_i y + c_i = 0, \quad i = \overline{1, N},$$

где параметры a_i и b_i выбираются так, чтобы нормальные векторы (a_i, b_i) были единичными и были направлены внутрь многоугольника.

Можно показать, что искомую окружность с наибольшим радиусом следует искать среди тех, которые касаются трех сторон многоугольника. Действительно, если окружность не касается ни одной из сторон, то можно сохранить ее центр, но увеличить радиус. Если касается лишь одной стороны, то можно сместить центр в направлении перпендикулярном соответствующей прямой (от нее), где снова можно будет увеличить радиус. Если окружность касается двух сторон, то движение можно осуществить в направлении биссектрисы угла, образованного этими сторонами (в случае если это не смежные стороны, то угол образуется их продолжениями), где снова исчезнет касание и вновь можно будет увеличить радиус. Исключение составляет случай, когда это две параллельные стороны. В этом случае мы не сможем никак “оторваться” от них, но можем просто двигаться в направлении, определяемом этими сторонами до тех пор, пока не коснемся какой-то третьей стороны. Следовательно существует окружность с максимальным радиусом, касающаяся каких-то трех сторон.

Однако по трем касательным однозначно определяется окружность (если только они не пересекаются в одной точке или среди них есть совпадающие прямые, но это частный случай предыдущего условия). Параметры окружности (координаты центра (x, y) и радиус R) касающейся, например, первых

трех сторон, определяются из системы трех линейных уравнений:

$$\begin{aligned} a_1x + b_1y + c_1 &= R, \\ a_2x + b_2y + c_2 &= R, \\ a_3x + b_3y + c_3 &= R, \end{aligned}$$

которые имеют место в силу соответствующей договоренности относительно выбора коэффициентов уравнений. Построив окружность по трем прямым, необходимо проверить, что эта окружность лежит целиком внутри многоугольника. Это так если расстояние от ее центра до каждой стороны не превосходит R , или в терминах аналитической геометрии:

$$a_i x + b_i y + c_i \leq R, \quad i = \overline{1, N}, (*)$$

Таким образом, перебирая все тройки сторон, и проверяя, лежит ли определяемая ими окружность внутри многоугольника, можно выбрать наилучшую из всех окружностей, прошедших эту проверку. Такой алгоритм будет работать за время $O(N^4)$, что укладывается в отведенное время для варианта Junior.

Для получения решения в варианте High рассмотрим систему неравенств (*). Она определяет в трехмерном пространстве некоторый многогранник, а исходный многоугольник является его сечением плоскостью $R = 0$. Однако количество вершин этого многогранника может быть весьма велико, поэтому строить его целиком не имеет смысла. Зафиксируем некоторое значение R , тогда неравенства (*) определяют N полуплоскостей на плоскости (x, y) . Построим с помощью алгоритма из лекции за $O(N \log N)$ пересечение этих полуплоскостей. Если оно окажется непустым, то может быть построена вписанная окружность радиуса R в любой точке этого пересечения. Если же пересечение пусто, то не существует вписанных окружностей радиуса R . Остается применить бинарный поиск по радиусу. Следует отметить, что поскольку в нашем варианте прямые уже отсортированы по углу, то можно предложить алгоритм для нахождения пересечения полуплоскостей с асимптотикой $O(N)$, и тогда общая сложность решения будет иметь порядок $O(N \log(M/\varepsilon))$. Однако и стандартный алгоритм при хорошей реализации даст возможность уложиться в отведенное время.

Задача S. Разделяющая прямая (высшая лига)

Вход: **stdin**
 Выход: **stdout**
 Ограничение по времени: 1 с
 Ограничение по памяти: 64 Мб

Задано множество, состоящее из N точек на плоскости. Прямая называется разделяющей по отношению к данному множеству, если найдутся две точки этого множества, лежащие в разных полуплоскостях относительно этой прямой (но не на самой прямой). Ваша задача — определить для каждой из заданных прямых, является ли она разделяющей или нет.

Ограничения

$0 \leq N, M \leq 10^5$, все координаты — целые числа, не превосходящие 10^9 по абсолютной величине.

Формат входного файла

В первой строке задается целое число N . В каждой из последующих N строк содержатся по два целых числа x_i, y_i , определяющие координаты соответствующей точки множества. В $(N + 2)$ -ой строке задается число M , а в последующих M строках — по 4 числа X_1, Y_1, X_2, Y_2 , где (X_1, Y_1) и (X_2, Y_2) — две различные точки на соответствующей прямой.

Формат выходного файла

Выведите M строк, каждая из которых определяет результат для соответствующей прямой. Если прямая является разделяющей, строка должна содержать два числа — номера каких-либо точек множества, лежащих в разных полуплоскостях относительно прямой. Если же прямая — неразделяющая, строка должна содержать одно число 0.

Пример

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

Задача Т. Разделяющая прямая (юниорская лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Задано множество, состоящее из N точек на плоскости. Прямая называется разделяющей по отношению к данному множеству, если найдутся две точки этого множества, лежащие в разных полуплоскостях относительно этой прямой (но не на самой прямой). Ваша задача — определить для каждой из заданных прямых, является ли она разделяющей или нет.

Ограничения

$0 \leq N, M \leq 10^4$, все координаты — целые числа, не превосходящие 10^9 по абсолютной величине.

Формат входного файла

В первой строке задается целое число N . В каждой из последующих N строк содержатся по два целых числа x_i, y_i , определяющие координаты соответствующей точки множества. В $(N + 2)$ -ой строке задается число M , а в последующих M строках — по 4 числа X_1, Y_1, X_2, Y_2 , где (X_1, Y_1) и (X_2, Y_2) — две различные точки на соответствующей прямой.

Формат выходного файла

Выведите M строк, каждая из которых определяет результат для соответствующей прямой. Если прямая является разделяющей, строка должна содержать два числа — номера точек множества, лежащих в разных полуплоскостях относительно прямой. Если же прямая — неразделяющая, нужно вывести одно число 0.

Пример

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

Разбор задачи Т. Разделяющая прямая

Проверку того, является ли некоторая прямая разделяющей, можно выполнить за время $O(N)$. Для этого для каждой точки можно определить по какую сторону от прямой она лежит. Удобно использовать для этого аппарат векторных произведений. Если величина $\overrightarrow{AB} \times \overrightarrow{AX}$ является положительной, то точка X лежит в левой полуплоскости относительно прямой AB , на которой выбрано направление \overrightarrow{AB} . Если указанное векторное произведение отрицательно, то X лежит по правую сторону от прямой AB , если же в точности равно нулю, то точки A, B и X лежат на одной прямой.

Такого варианта проверки будет достаточно для того, чтобы решить задачу в варианте Junior, а общее время работы будет иметь порядок $O(MN)$.

Для того, чтобы уметь отвечать для каждой прямой быстрее чем за $O(N)$ очевидно нужна какая-то предварительная обработка заданного множества точек.

Прежде всего заметим, что прямая будет разделяющей тогда и только тогда, когда она является разделяющей и для выпуклой оболочки этого множества. Таким образом, после прочтения всех точек достаточно оставить лишь вершины входящие в выпуклую оболочку, вычислив ее за время порядка $O(N \log N)$.

Теперь наша задача заключается в следующем — для заданной прямой необходимо определить пересекает ли она заданный выпуклый многоугольник. Это может быть сделано следующим образом. Построим нормальный вектор к данной прямой, который задаст некоторое направление. Очевидно, в многоугольнике есть самая левая и самая правая точки в таком направлении. Если они лежат в одной и той же полуплоскости относительно прямой (или одна из них — непосредственно на прямой), то отсюда будет следовать, что и все остальные точки лежат в той же полуплоскости, и прямая не является разделяющей. Если же эти точки лежат в разных полуплоскостях, то именно их и можно предъявить в качестве сертификата разделения.

Остается научиться находить эти крайние точки для направления, определяемого некоторым полярным углом. Заметим, что точка A_i будет самой дальней в направлении, которое находится между направлениями вектора $\overrightarrow{A_i A_{i+1}}$ и $\overrightarrow{A_{i-1} A_i}$ (если оболочка построена с учетом обхода против часовой стрелки). Поэтому если мы запишем все полярные углы, которые образуют последовательные векторы выпуклой оболочки, то сможем находить самые дальние точки в требуемом направлении за время $O(\log N)$, используя технику бинарного поиска.

Разумеется для повышения точности вычислений имеет смысл использовать не полярный угол, а осуществлять поиск, используя в качестве критерия

векторное (или скалярное) произведение.

Итак, описанный алгоритм тратит время $O(N \log N)$ на предобработку, и по $O(\log N)$ для каждого из M запросов. Таким образом, суммарная сложность будет иметь порядок $O((M + N) \log N)$.

Задача U. Стингангуляция (высшая лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Пусть задана некоторая строка $s_1 s_2 \dots s_l$. Предполагаем, что она циклическая, то есть за s_l следует символ s_1 . Будем называть стингангуляцией этой строки ее разбиение на три последовательные подстроки (уже не циклические, обозначим их соответственно a , b и c), для каждой из которых выполнено неравенство треугольника, то есть

$$a + b > c, \quad b + c > a, \quad c + a > b,$$

где знаком “+” понимается как обычно для строк их конкатенация.

Ваша задача — найти для заданной строки количество различных ее стингангуляций. Стингангуляции, отличающиеся лишь обозначениями последовательных подстрок, не считаются различными.

Ограничения

Длина строки s находится в диапазоне от 3 до 2014.

Формат входного файла

Единственная строка определяет строку s .

Формат выходного файла

Выведите одно число — количество различных стингангуляций строки s .

Пример

stdin	stdout
aaa	1
cbcccbcacb	2
stringangulation	0

Задача V. Стингангуляция (юниорская лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

Пусть задана некоторая строка $s_1 s_2 \dots s_l$. Предполагаем, что она циклическая, то есть за s_l следует символ s_1 . Будем называть стингангуляцией этой строки ее разбиение на три последовательные подстроки (уже не циклические, обозначим их соответственно a , b и c), для каждой из которых выполнено неравенство треугольника, то есть

$$a + b > c, b + c > a, c + a > b,$$

где знаком “+” понимается как обычно для строк их конкатенация.

Ваша задача — найти для заданной строки количество различных ее стингангуляций. Стингангуляции, отличающиеся лишь обозначениями последовательных подстрок, не считаются различными.

Ограничения

Длина строки s находится в диапазоне от 3 до 100.

Формат входного файла

Единственная строка определяет строку s .

Формат выходного файла

Выведите одно число — количество различных стингангуляций строки s .

Пример

stdin	stdout
aaa	1
cbccbcbc	2
stringangulation	0

Разбор задачи V. Стингангуляция

Непосредственно проверяется, что ни одна из подстрок не может быть пустой. Зафиксируем обозначения разрезанных подстрок таким образом, чтобы символ s_l относился к подстроке c . Тогда очевидно, что любая тройка чисел (i, j, k) , удовлетворяющая неравенствам $1 \leq i < j < k \leq l$, определяет

некоторое разбиение исходной строки s на подстроки: подстрока a начинается с i -ым символом и заканчивается $(j - 1)$ -ым, подстрока b начинается j -ым символом и заканчивается $(k - 1)$ -ым, подстрока c начинается k -ым символом и заканчивается $(i - 1)$ -ым. Всего таких разбиений будет порядка $O(l^3)$. Остается лишь выяснить сколько из таких разбиений удовлетворяют неравенствам для стрингангуляции. Очевидно для этого нужно сравнить три пары строк: $s[i : j - 1]$ с $s[j : i - 1]$, $s[j : k - 1]$ с $s[k : j - 1]$, $s[k : i - 1]$ с $s[i : k - 1]$. Каждое из этих сравнений может быть очевидным образом выполнено за $O(l)$. Таким образом, имеем алгоритм с асимптотикой $O(l^4)$, которой достаточно для варианта Junior.

Мы можем ускорить работу алгоритма, если заранее предподсчитаем для каждой пары (i, j) минимальное неотрицательное целое k такое, что символ s_{i+k} отличается от символа s_{j+k} . Если для всех k от 0 до $l - 1$ сохраняется равенство, то формально k будет бесконечным, но нам достаточно будет считать, что оно равно l . Обозначим для заданных i и j соответствующее значение k через $leq(i, j)$. Это может быть сделано прямым алгоритмом за $O(l^3)$ операций, или с помощью идеи динамического программирования за время $O(l^2)$ (зная значение $leq(i, j)$, легко получить значение $leq(i - 1, j - 1)$, оно будет либо на единицу больше, либо равно нулю, в зависимости от того, равны ли символы s_i и s_j или нет.).

Тогда сравнение двух подстрок $s[i : i + l_i - 1]$ и $s[j : j + l_j - 1]$ может быть выполнено за время $O(1)$. Для этого достаточно рассмотреть чему равно значение $k = leq(i, j)$. Если оно строго меньше длин подстрок (l_i и l_j), то результат их сравнения определяется результатом сравнения символов s_{i+k} и s_{j+k} , в противном случае — результатом сравнения длин подстрок.

Тем самым мы уменьшили сложность работы алгоритма до $O(l^3)$, чего все равно пока еще недостаточно для варианта High.

Автор задачи искренне верит в существование алгоритма с более хорошей асимптотикой временной сложности, но, к сожалению, получить его пока не удалось.

Все, что дальше предлагается сделать, приведет к уменьшению константы при l^3 примерно в 16 раз. Для этого для каждого i заведем два множества:

$$L_i = \{j | s[i : j - 1] < s[j : i - 1]\},$$

$$G_i = \{j | s[i : j - 1] > s[j : i - 1]\}.$$

Их все очевидно можно построить за $O(l^2)$. Тогда мы можем для каждого i перебирать все значения $j > i$ из множества L_i . И тогда нужно будет найти количество таких $k > j$ в пересечении множеств L_j и G_i . Если представить множества в виде битовых массивов, каждый бит которого отвечает за наличие некоторого элемента в соответствующем множестве, объединив по 16 бит

в одном значении целочисленного типа соответствующей размерности, у нас появится возможность существенно ускорить пересечение двух множеств (за счет побитовых операций). Останется лишь посчитать количество единичных битов в каждом из $l/16$ значениях 16-битового целого типа. Для каждого такого значения это может быть сделано за $O(1)$, если предварительно предподсчитать количества бит для каждого из значений от 0 до $2^{16} - 1$. Указанное решение будет давать результат уже за вполне приемлемое время и в варианте ограничений High.

Задача W. Построение куба (высшая лига)

Вход:	<code>stdin</code>
Выход:	<code>stdout</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

В трехмерном пространстве задана плоскость уравнением $Ax + By + Cz + D = 0$. Требуется построить куб, вершины которого удалены от этой плоскости на заданные расстояния d_1, d_2, \dots, d_8 .

Ограничения

Все значения — вещественные, не превышают 10^4 по абсолютной величине и имеют не более двух знаков в дробной части. Хотя бы одно из чисел A, B и C отлично от нуля, все значения d_i — неотрицательны.

Формат входного файла

В первой строке задаются коэффициенты уравнения плоскости A, B, C и D , а во второй строке расстояния — d_1, d_2, \dots, d_8 .

Формат выходного файла

Выведите 8 строк, каждая из которых будет содержать по три числа — координаты x, y и z соответствующей вершины куба. Первая выведенная точка должна находиться на расстоянии d_1 от плоскости (с точностью до 10^{-5}), вторая — на расстоянии d_2 и т.д.

Пример

stdin	stdout
1.0 1.0 1.0 1.0	0.244017 0.244017
1.0 3.0 2.0 4.0 2.0 3.0 3.0	0.244017
2.0	1.976068 1.976068
	0.244017
	1.976068 0.244017
	0.244017
	1.976068 1.976068
	1.976068
	0.244017 1.976068
	0.244017
	1.976068 0.244017
	1.976068
	0.244017 1.976068
	1.976068
	0.244017 0.244017
	1.976068
-1 2 -3 6	Impossible
0 0 0 0 0 0 0 0	

Задача X. Построение квадрата (юниорская лига)

Вход: **stdin**
Выход: **stdout**
Ограничение по времени: 1 с
Ограничение по памяти: 64 Мб

На плоскости задана прямая уравнением $Ax + By + C = 0$. Требуется построить квадрат, вершины которого удалены от этой прямой на заданные расстояния d_1, d_2, d_3, d_4 .

Ограничения

Все значения — вещественные, не превышают 10^4 по абсолютной величине и имеют не более двух знаков в дробной части. Хотя бы одно из чисел A и B отлично от нуля, все значения d_i — неотрицательны.

Формат входного файла

В первой строке задаются коэффициенты уравнения прямой A, B, C , а во второй строке расстояния — d_1, d_2, d_3, d_4 .

Формат выходного файла

Выведите 4 строки, каждая из которых будет содержать по два числа — координаты x и y соответствующей вершины квадрата. Первая выведенная точка должна находиться на расстоянии d_1 от прямой (с точностью до 10^{-5}), вторая — на расстоянии d_2 и т.д.

Пример

stdin	stdout
1.0 2.0 3.0	-0.152786 -0.305573
1.0 2.0 3.0 4.0	2.083282 -0.305573 -0.152786 1.930495 2.083282 1.930495
-1 2 -3 6	Impossible
0 0 0 0 0 0 0 0	

Разбор задачи X. Построение куба

Основная часть задачи была разобрана на лекции зимней школы 2008 года. Приведем здесь для полноты то же решение.

Предположим сначала, что все точки лежат по одну сторону от плоскости (или часть из них на самой плоскости).

Обозначим наименьшее из расстояний заданных расстояний d_0 . Предположим, что можно построить нужный куб. Пусть проекции на перпендикуляр к плоскости ребер, выходящих из ближайшей к плоскости вершины, равны x , y и z соответственно. Тогда среди заданных расстояний должны быть еще d_0+x , d_0+y и d_0+z . Упорядочим их так, чтобы выполнялось $x \leq y \leq z$. Остальные ребра должны быть параллельны и равны первым трем. А поскольку проекции параллельных и равных между собой отрезков на одну и ту же прямую равны, между собой, то должны быть еще расстояния $-d+x+y$, $d+x+z$, $d+y+z$, $d+x+y+z$. Итак, если можно построить куб, то расстояния должны быть представимы в виде d_0 , d_0+x , d_0+y , d_0+z , d_0+x+y , d_0+x+z , d_0+y+z , $d_0+x+y+z$, где $d \geq 0$ и $0 \leq x \leq y \leq z$. Если это не так, то куб построить невозможно.

Если бы требовалось построить не куб, а параллелепипед, то можно было бы взять 4 произвольные различные точки на данной плоскости. Затем сдвинуть их на расстояния $d_i(A^2 + B^2 + C^2)^{-1/2}(A, B, C)$ ($i = \overline{1, 4}$). Таким образом, мы бы получили координаты ближней вершины и еще трех смежных с ней вершин. Дальше все вершины определяются однозначно и формулы для них очевидны.

Несложно решить аналогичную задачу для квадрата и прямой на плоскости. Сторона этого квадрата будет точно определяться расстояниями d и будет равна $\sqrt{x^2 + y^2}$, и сам квадрат определяется однозначно с точностью до сдвига вдоль прямой и симметрии относительно прямой, перпендикулярной данной. Обозначим $l = \sqrt{A^2 + B^2}$, тогда вектор единичной нормали \vec{n} к прямой $Ax + By + C = 0$ будет иметь вид $\vec{n} = l^{-1}(A, B)$, а направляющий вектор прямой $-\vec{\tau} = l^{-1}(B, -A)$.

Найдем самую ближнюю точку A к прямой (которая находится на расстоянии d_0). Будем искать ее в виде $A = a_n \vec{n} + a_\tau \vec{\tau}$. Тогда получим, что a_n будет равно $C \pm l^{-1}d_0$, а a_τ — произвольное число. Выбор знака плюс или минус обеспечивается выбором полуплоскости в которой будет строиться квадрат, а величина a_τ определяет сдвиг вдоль прямой. Теперь для получения следующих точек (находящихся на расстоянии x и y) нужно будет добавить к A вектор $x\vec{n} \pm y\vec{\tau}$ и $y\vec{n} \mp x\vec{\tau}$ (выбор знаков должен быть сделан так, чтобы они оказались противоположными), это определяет два варианта, один из которых является симметричным по отношению к другой относительно некоторой прямой, параллельной вектору \vec{n} . Последняя же точка теперь определится однозначно — нужно будет к A добавить оба указанных вектора.

Задача о квадрате полностью решена. Можно заметить, что мы фактически ввели на исходной плоскости новый базис $\vec{n}, \vec{\tau}$, в котором уравнений прямой и выражения для координат точек оказались наиболее простыми.

Вернемся снова к задаче о построении куба. С помощью аппарата аналитической геометрии и линейной алгебры (задача о нахождении ортогональной матрицы удовлетворяющей уравнению $Xa = b$) можно доказать, что для исходной задачи при выполнении найденного нами условия всегда можно построить куб. Укажем способ построения.

Запишем снова $l = \sqrt{A^2 + B^2 + C^2}$ и единичный нормальный к плоскости $\vec{n} = l^{-1}(A, B, C)$. Пусть векторы, выходящие из ближайшей вершины, имеют вид $\vec{X} = (x_1, x_2, x_3), \vec{Y} = (y_1, y_2, y_3)$ и $\vec{Z} = (z_1, z_2, z_3)$. Их проекции на перпендикуляр к плоскости соответственно равны x, y и z . Тогда из условия ортогональности и равенства по модулю векторов $\vec{X}, \vec{Y}, \vec{Z}$ имеем систему: $\vec{X} \cdot \vec{n} = x, \vec{Y} \cdot \vec{n} = y, \vec{Z} \cdot \vec{n} = z, \vec{X} \cdot \vec{Y} = \vec{X} \cdot \vec{Z} = \vec{Y} \cdot \vec{Z} = 0, \vec{X}^2 = \vec{Y}^2 = \vec{Z}^2 = a^2$, где a — длина стороны искомого куба. Если мы решим данную систему, то дальше построить куб не составит труда.

Разделим $\vec{X}, \vec{Y}, \vec{Z}, x, y, z$ на a , тогда уравнения примут вид: $\vec{X}' \cdot \vec{n} = x', \vec{Y}' \cdot \vec{n} = y', \vec{Z}' \cdot \vec{n} = z', \vec{X}' \cdot \vec{Y}' = \vec{X}' \cdot \vec{Z}' = \vec{Y}' \cdot \vec{Z}' = 0, \vec{X}'^2 = \vec{Y}'^2 = \vec{Z}'^2 = 1$, где штрихами обозначены величины, которые получаются после деления. Отсюда видно, что \vec{X}', \vec{Y}' и \vec{Z}' образуют ортонормированный базис трехмерного пространства. Следовательно, если записать эти векторы в строки некоторой

матрицы, то получим ортогональную матрицу H . Обозначим $\vec{p} = (x', y', z')$. Из первых трех равенств следует, что $H\vec{n} = \vec{p}$. Таким образом, \vec{p} – это образ вектора \vec{n} , к которому применен ортогональный оператор H . Как известно, ортогональный оператор является поворотом (возможно в композиции с симметрией), поэтому векторы \vec{n} и \vec{p} имеют одну и ту же длину 1. Отсюда следует, что $x'^2 + y'^2 + z'^2 = 1$ или, если переписать в исходных переменных, $x^2 + y^2 + z^2 = a^2$. Таким образом длина ребра уже уже определена.

Теперь осталось найти хотя бы одну (любую) ортогональную матрицу, удовлетворяющую соотношению $H\vec{n} = \vec{p}$, то есть любое преобразование, сохраняющее длины, которое переведет вектор \vec{n} в вектор \vec{p} . Если $\vec{n} = \vec{p}$, то можно взять тождественный оператор $H = E$. Если же $\vec{n} \neq \vec{p}$, то можно взять, например, преобразование симметрии относительно некоторой плоскости. Эта плоскость должна быть перпендикулярной вектору $\vec{p} - \vec{n}$ и проходить через начало координат. Ее уравнение можно записать в виде $(\vec{p} - \vec{n}) \cdot \vec{t} = 0$, где \vec{t} – точка пространства. Тогда наше преобразование можно записать в виде формулы $H\vec{t} = \vec{t} - 2|\vec{p} - \vec{n}|^{-1}((\vec{p} - \vec{n}) \cdot \vec{t})$. Для того, чтобы получить саму матрицу, нужно применить данное преобразование к базисным векторам. Остается только получить координаты ближней вершины A . Опять же будем искать ее в виде $a_n\vec{n}$. Проделав все преобразования, получим, что $a_n = D \pm l^{-1}d_0$. Откладывая от найденной точки векторы $a\vec{X}'$, $a\vec{Y}'$, $a\vec{Z}'$, где \vec{X}' , \vec{Y}' , \vec{Z}' – соответствующие строки матрицы H , получим смежные с A вершины куба. Остальные вершины можно будет достроить уже однозначно.

Вернемся теперь к исходной формулировке. Когда не утверждается, что все точки лежат по одну сторону от плоскости. Тогда переберем все 2^8 вариантов подмножеств точек, которые будут лежат в другом полупространстве и запустим предыдущее решение, взяв в качестве расстояний до точек, “переведенных” в другое полупространство, соответствующие отрицательные числа. Если для какого-то из вариантов получится куб, то его и выведем в качестве ответа.

Можно заметить еще, что достаточно проверить лишь 2^7 вариантов, поскольку есть пары симметричных вариантов относительно исходной плоскости (например, варианту, когда все вершины лежат в одном полупространстве, соответствует симметричный вариант, в котором все вершины лежат в другом полупространстве). Поэтому одну из вершин можно зафиксировать как лежащую всегда в одном и том же полупространстве.