

Министерство образования и науки Украины



Яndex



# Зимняя школа по программированию

Харьков, ХНУРЭ

2011



---

## Оглавление

День первый. Контест Пака Станислава Олеговича . . . . .	8
Об авторе... . . . . .	8
Теоретический материал. 3D-геометрия . . . . .	8
Задачи и разборы . . . . .	12
Задача А. Касательные к сферам . . . . .	12
Задача В. Расстояние между отрезками . . . . .	13
Задача С. Конусы . . . . .	14
Задача Д. Снайпер . . . . .	15
Задача Е. Выпуклая оболочка 3D - 1 . . . . .	17
Задача F. Прямые . . . . .	20
Задача G. Взаимное расположение прямых . . . . .	22
Задача Н. Выпуклая оболочка 3D - 2 . . . . .	23
Задача I. Выпуклая оболочка 3D - 3 . . . . .	25
Задача J. Расстояние от точки до отрезка . . . . .	26
Задача K. Цилиндр . . . . .	27
Задача L. Разрежем арбуз . . . . .	28
День второй. Контест Жукова Дмитрия Ивановича . . . . .	30
Об авторе... . . . . .	30
Теоретический материал. Алгоритм Кацаубы. Быстрое преобразование Фурье . . . . .	31
Задачи и разборы . . . . .	36
Задача A. ДНК роботов . . . . .	36
Задача B. Треугольник . . . . .	38
Задача C. Монеты . . . . .	40
Задача D. Уравнение . . . . .	42
Задача E. Разворот битов . . . . .	43
Задача F. АВЛ-деревья . . . . .	45
Задача G. Многочлен . . . . .	47
Задача H. Раздвоение . . . . .	48
Задача I. Уравнение 2 . . . . .	50
Задача J. Разворот битов 2 . . . . .	51
Задача K. Дуэль . . . . .	53
Задача L. Произведение . . . . .	54
Задача M. Сфера . . . . .	55
День третий. Контест Дворкина Михаила Эдуардовича . . . . .	57
Об авторе... . . . . .	57
Теоретический материал. Динамическое программирование по профилю и по изломанному профилю . . . . .	57
Задачи и разборы . . . . .	62

---

Задача А. Треугольный король . . . . .	62
Задача В. Палатка . . . . .	63
Задача С. Чемпионат по грибному спорту . . . . .	64
Задача Д. Виннимобиль . . . . .	67
Задача Е. Схемы рифмовки . . . . .	68
Задача F. Шахтеры . . . . .	70
Задача G. Подарок Пятачку . . . . .	72
Задача Н. Играчная кость . . . . .	73
Задача I. Пятизвездочная задача . . . . .	74
Задача J. Прочные замощения . . . . .	76
Задача К. Гамильтонов трубопровод . . . . .	78
Задача L. Шестиугольник и ромбические домино . . . . .	80
Задача М. Симпатичные узоры 2 . . . . .	81
День четвертый. День Киевского Политехнического Института . . . . .	83
Об авторах... . . . . .	83
Теоретический материал. Вероятностные структуры данных . . . . .	84
Задачи и разборы . . . . .	90
Задача А. Инопланетяне . . . . .	90
Задача В. Конфеты . . . . .	91
Задача С. Раскраска . . . . .	93
Задача D. Сортировка вагонов . . . . .	94
Задача Е. Максимальный поток . . . . .	96
Задача F. Митохондриальная ДНК . . . . .	97
Задача G. Кратные запросы . . . . .	99
Задача Н. Ориентирование . . . . .	100
Задача I. Гонки . . . . .	102
Задача J. TR3N . . . . .	104
Задача К. Хамелеоны . . . . .	107
Задача L. Площадь кольца . . . . .	108
День пятый. Контест Копелиовича Сергея Владимировича . . . . .	110
Об авторе... . . . . .	110
Теоретический материал. Перебор с отсечениями . . . . .	110
Задачи и разборы . . . . .	119
Задача А. Карточки . . . . .	119
Задача В. Функция . . . . .	120
Задача С. Пути на доске . . . . .	121
Задача D. Тестирование шаффл-машин . . . . .	122
Задача Е. Белка и бамбук . . . . .	123
Задача F. Снеговики . . . . .	125
Задача G. Вершинное покрытие . . . . .	126
Задача Н. Праздничные дни . . . . .	128

---

---

Задача I. Самый длинный путь . . . . .	130
Задача J. Сумма не без разнообразия . . . . .	132
Задача K. Валидатор к игре в тетрис . . . . .	133
Задача L. Возьми себе за правило — летай всегда GraphAero! . . . . .	135
Задача M. Тест к задаче про Клики . . . . .	137
Задача N. Задача про Клики . . . . .	139
Задача O. Connect and Disconnect . . . . .	141
Задача P. Кривые зеркала . . . . .	142
Задача Q. Сортировка вручную . . . . .	144
Задача R. СНМ . . . . .	146
День шестой. Контест Виталия Неспирного и Антона Лунёва . . . . .	148
Об авторах... . . . . .	148
Теоретический материал. Близость точек на плоскости. Диаграмма Вороного . . . . .	150
Задачи и разборы . . . . .	158
Задача A. Количество линий . . . . .	158
Задача B. Восстановление количества очков . . . . .	160
Задача C. Как побить все рекорды . . . . .	162
Задача D. Секретный уровень . . . . .	164
Задача E. Поля сражений . . . . .	165
Задача F. Взлом таблицы рекордов . . . . .	167
Задача G. Горная гряда . . . . .	169
Задача H. Постройка склада . . . . .	171
Задача I. Сила героя . . . . .	173
Задача J. Караваны . . . . .	176
Задача K. Кто ходит в гости по утрам . . . . .	177
Задача L. Принц или самозванец . . . . .	179
Задача M. Что? Где? Когда? . . . . .	181
Задача N. Укладка плит . . . . .	183
Задача O. Прохождение коридора . . . . .	185
Задача P. Игра . . . . .	187
Задача Q. Гамильтонов цикл . . . . .	188
Задача R. Очень дружная группа . . . . .	190
Задача S. Деревья с нечетным числом независимых множеств . . . . .	193
Задача T. Максимальная степень простого . . . . .	195
Задача U. Минимальная степень простого . . . . .	197
День седьмой. Контест Эльдара Богданова и Ивана Метельского . . . . .	199
Об авторах... . . . . .	199
Теоретический материал. Ориентированные графы. Задача 2-SAT. . . . .	200
Задачи и разборы . . . . .	225
Задача A. Anniversary . . . . .	225

---

Задача B. Trade . . . . .	228
Задача C. Grid . . . . .	231
Задача D. Drawing . . . . .	233
Задача E. Empire . . . . .	235
Задача F. Reform . . . . .	238
Задача G. Constellation . . . . .	242
Задача H. Presents . . . . .	244
Задача I. ICPC . . . . .	247
Задача J. Numbers . . . . .	255
Задача K. English . . . . .	256
Задача L. Diamond . . . . .	258
Задача M. Puzzle . . . . .	260
День восьмой. Контест Митричева Петра Игоревича . . . . .	262
Об авторе... . . . . .	262
Теоретический материал. Задачи “от $L$ до $R$ ” . . . . .	263
Задачи и разборы . . . . .	266
Задача A. Automatic Input Verifier . . . . .	266
Задача B. Friendly Points . . . . .	270
Задача C. Expanding Lake . . . . .	271
Задача D. Octahedron And Dominoes . . . . .	272
Задача E. Tiny Puzzle . . . . .	275
Задача F. Circular Island . . . . .	276
Задача G. Traffic Jam . . . . .	278
Задача H. Number Permutations . . . . .	279
Задача I. Perspective . . . . .	280
Задача J. Enjoy Arithmetic Progressions . . . . .	282
Задача K. Completely Non-zero Determinant . . . . .	284
Задача L. Fast Typing . . . . .	285
Задача M. XYZ 2009 . . . . .	286
Задача N. Greatest Greatest Common Divisor . . . . .	289
Самое короткое решение: Distinct Substrings . . . . .	290
День девятый. Контест Гольдштейна Виталия Борисовича . . . . .	292
Об авторе... . . . . .	292
Теоретический материал. Применение обхода в глубину . . . . .	293
Задачи и разборы . . . . .	298
Задача A. Цветные волшебники – 2 . . . . .	298
Задача B. Граф операций . . . . .	300
Задача C. Регулярное паросочетание . . . . .	301
Задача D. Авиаперелеты . . . . .	303
Задача E. Противопожарная безопасность . . . . .	304
Задача F. Островные государства . . . . .	306

---

---

Задача G. Король . . . . .	308
Задача Н. Каток . . . . .	309
Задача I. Предок . . . . .	311
Задача J. Неизбежность . . . . .	312
Задача K. Мосты . . . . .	313
Задача L. Цветные волшебники . . . . .	314
Задача M. Точки сочленения . . . . .	316
Задача N. Почтальон . . . . .	317
Задача O. Конденсация графа . . . . .	319
Задача P. Топологическая сортировка . . . . .	320

## День первый (12.02.2011 г.) Конкурс Пака Станислава Олеговича

### Об авторе...

**Пак Станислав Олегович**, родился в 1986 году в городе Янгиюль республики Узбекистан. Учился в лингвистической гимназии №8 города Энгельса Саратовской области. С 8 класса участвовал в олимпиадах по физике, химии, математике, информатике, литературе. Закончил механико-математический факультет Саратовского государственного университета с красным дипломом, учится в аспирантуре СГУ по специальности 01.01.01 “Математический анализ”. Разработчик компании Яндекс. Интересуется психологией.



Основные достижения:

- первое место на областной олимпиаде по математике (Саратов, 2004);
- второе место в Южном четвертьфинале NEERC 2006 и 2007 года;
- чемпион России NEERC ACM 2008;
- золотая медаль ACM ICPC World Finals 2009 (Стокгольм).

## Теоретический материал. 3D-геометрия

### Определения

Будем отождествлять точки и вектора. Пусть точки  $a, b$  в пространстве  $\mathbb{R}^3$  являются концами отрезка  $s$ . Тогда отрезок описывается множеством  $\{a + (b - a) \cdot t\}, t \in [0, 1]$ . Здесь операции сложения и вычитания определяются как над векторным пространством, умножение на число — скаляра на вектор. Множество точек прямой, проходящей через две точки  $a$  и  $b$  описывается как множество  $\{a + (b - a) \cdot t\}, t \in \mathbb{R}$ .

Определим операцию скалярного умножения векторов  $v, u$  как  $v \cdot u = \sum_{i=1}^3 v_i \cdot u_i$ , где  $u_i, v_i$  — их координаты, часто не будем различать  $v_1$  и  $v.x$ ,  $v_2$  и  $v.y$ ,  $v_3$  и  $v.z$ .

Определим операцию векторного умножения векторов  $v, u$  как

$$v \times u = \det \begin{pmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ v.x & v.y & v.z \\ u.x & u.y & u.z \end{pmatrix} = 0,$$

где  $\mathbf{i}, \mathbf{j}, \mathbf{k}$  — орты декартовой системы координат, то есть результатом этой операции будет вектор  $r$ :

$$r.x = \det \begin{pmatrix} v.y & v.z \\ u.y & u.z \end{pmatrix}, r.y = -\det \begin{pmatrix} v.x & v.z \\ u.x & u.z \end{pmatrix}, r.z = \det \begin{pmatrix} v.x & v.y \\ u.x & u.y \end{pmatrix}.$$

Векторное произведение равно 0, если и только если  $v, u$  — коллинеарны. Результатом векторного произведения является вектор, ортогональный  $v, u$  и направленный так, что если смотреть из его конца, то вектора  $v$  и  $u$  будут ориентированы против часовой стрелки.

По трем точкам  $a, b, c \in \mathbb{R}^3$ , если они не принадлежат одной прямой, можно построить плоскость  $\alpha$ . Рассмотрим вектора  $v_1 = b - a, v_2 = c - a$  и их векторное произведение  $v_p = v_1 \times v_2$ . Пусть  $p$  — точка, принадлежащая плоскости  $\alpha$ . Тогда вектор  $p - a$  ортогонален  $v_p$ . Верно и обратное. Поэтому условие  $(p - a) \cdot ((b - a) \times (c - a)) = 0$  является критерием того, что точка  $p$  принадлежит  $\alpha$ . Таким образом, уравнение плоскости описывается как

$$\det \begin{pmatrix} x - a.x & y - a.y & z - a.z \\ b.x - a.x & b.y - a.y & b.z - a.z \\ c.x - a.x & c.y - a.y & c.z - a.z \end{pmatrix} = 0.$$

Очевидно, если записать уравнение плоскости в виде  $Ax + By + Cz + D = 0$ , то коэффициенты вычисляются следующим образом:

$$A = \det \begin{pmatrix} (b - a).y & (b - a).z \\ (c - a).y & (c - a).z \end{pmatrix}, B = -\det \begin{pmatrix} (b - a).x & (b - a).z \\ (c - a).x & (c - a).z \end{pmatrix},$$

$$C = \det \begin{pmatrix} (b - a).x & (b - a).y \\ (c - a).x & (c - a).y \end{pmatrix}, D = -\det \begin{pmatrix} a.x & a.y & a.z \\ (b - a).x & (b - a).y & (b - a).z \\ (c - a).x & (c - a).y & (c - a).z \end{pmatrix}$$

## Приложения

Часто в задачах приходится определять взаимное расположение геометрических объектов и расстояние между ними. Рассмотрим эти вопросы на примере простейших геометрических фигур в пространстве.

Как найти расстояние между прямыми?

Пусть  $a + (b - a) \cdot t_1, c + (d - c) \cdot t_2$  — параметрическое задание прямых. Расстояние между произвольными точками на прямых

$$r(t_1, t_2) = \sqrt{\sum_{i=1}^3 (a + (b - a) \cdot t_1 - (c + (d - c) \cdot t_2))_i}. \text{ Задача свелась к поиску минимума функции. Можно рассматривать ее квадрат. Аналогично для отрезков, только с ограничениями на параметр } t_1, t_2 \in [0, 1]. \text{ Знанное расстояние между точкой } p \text{ и плоскостью, заданной уравнением } Ax + By + Cz + D = 0, \text{ находится как } \frac{Ap.x + Bp.y + Cp.z + D}{\sqrt{A^2 + B^2 + C^2}}.$$

Пусть две плоскости описаны уравнениями  $A_1x + B_1y + C_1z + D_1 = 0, A_2x + B_2y + C_2z + D_2 = 0$ . Часто нужно уметь находить угол между плоскостями. Это равносильно рассмотрению угла между векторами  $(A_1, B_1, C_1)$  и  $(A_2, B_2, C_2)$ .

Так же бывает полезно уметь сортировать вектора, лежащие в одной плоскости, по полярному углу. Для этого можно зафиксировать один вектор  $v_0$  и разделить вектора на два множества: слева от  $v_0$  и справа от  $v_0$ . Можно брать векторное произведение  $v_0 \times v_c$  и проверять на сонаправленность с вектором  $v_c$ , здесь  $v_c$  — вектор ортогональный плоскости, в которой лежат вектора. Этот метод работает и для векторов, которые, например, являются сторонами с общим концом некоторого выпуклого многогранника.

## Элементы геометрии на сфере

Рассмотрим сферу  $S$  радиуса  $R$  с центром в начале координат.

## Некоторые факты

- Для точки на сфере существует единственная диаметрально ей противоположная точка на сфере. Таким образом, все точки сферы делятся на пары диаметрально противоположных точек.
- Прямой на сфере назовем замкнутую линию на сфере, лежащую в некоторой плоскости, проходящей через центр сферы. Через две точки, не являющиеся диаметрально противоположными, можно провести единственную прямую. Будем отождествлять прямую на сфере с плоскостью, в которой эта прямая лежит. Отрезком назовем непрерывную замкнутую часть прямой (дуга окружности).

- Две прямые на сфере либо совпадают, либо пересекаются ровно в двух диаметрально противоположных точках.
- Через точку можно провести единственную прямую, перпендикулярную данной  $L$ , за исключением случая, когда эта точка лежит на прямой, ортогональной к плоскости, ассоциированной с  $L$ . В этом случае, будет бесконечно много таких прямых. Примером такого случая является экватор и, например, северный полюс.
- Расстоянием между двумя точками на сфере является длина кратчайшей дуги прямой на сфере, проходящей через две точки. Окружность на сфере с центром в точке  $C$  называется множество точек, равноудаленных (в смысле сферического расстояния) от точки  $C$ . Очевидно, что окружность на сфере является окружностью на плоскости. Также очевидно, что кратчайший путь между парой точек принадлежит прямой, проходящей через эти точки.
- Пересечем на сфере две прямые  $l$  и  $m$ . Пусть они пересекаются в диаметрально противоположных точках  $p$  и  $q$ . Сфера таким образом разбивается на 4 области, каждая из которых называется двуугольником. Получаем пары конгруэнтных двуугольников. Каждый двухгольник определяется своим углом с точностью до изометрии. Площадь двухгольника с углом  $\alpha$  равна  $2R^2\alpha$ .
- Проведем три различных прямых. Они либо пересекаются в двух точках, разбивая поверхность сферы на двуугольники, либо попарно пересекаются в шести различных точках, которые разбиваются на пары диаметрально противоположных точек. Таким образом, сфера разбивается на 8 областей, каждая из которых ограничена тремя отрезками. Назовем такие области треугольниками. Обозначим углы треугольника  $\alpha, \beta, \gamma$ . Здесь угол рассматривается как угол между плоскостями, ассоциированными с прямыми, на которых лежат смежные стороны. Легко доказать, что площадь сферического треугольника равна  $(\alpha + \beta + \gamma - \pi)R^2$ .

## Задачи и разборы

### Задача А. Касательные к сферам

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

Вам даны три сферы в пространстве. Найдите все их общие касательные плоскости.

#### Формат входного файла

В первой строке  $M$  — количество тестов. Следующие  $3M$  строк описывают сами сферы. Каждая сфера описывается 4 числами: координатами центра и радиусом. Все числа в файле целые; известно, что все координаты от  $-500$  до  $+500$ , и радиус больше нуля и не превосходит 500. Центры сфер не совпадают и не лежат на одной прямой.

#### Формат выходного файла

Для каждого теста выведите  $K$  — количество различных касательных плоскостей. Следующие  $K$  строк описывают эти плоскости. Каждая строка — четыре целых числа  $A, B, C, D$ , описывающих плоскость  $Ax + By + Cz = D$ , такие, что  $A^2 + B^2 + C^2 = 1$ . Описания плоскостей могут быть выданы в любом порядке. Выводите числа с точностью не менее 5 знаков после десятичной точки.

#### Пример

<b>a.in</b>	<b>a.out</b>
1	4
0 0 0 1	0.00000000 1.00000000 0.00000000 1.00000000
0 2 0 1	1.00000000 0.00000000 0.00000000 1.00000000
2 0 0 1	0.00000000 0.00000000 1.00000000 -1.00000000
	0.00000000 0.00000000 1.00000000 1.00000000

### Разбор задачи А. Касательные к сферам

Рассмотрим некоторую касательную плоскость, заданную нормализованным уравнением  $Ax + By + Cz + D = 0, A^2 + B^2 + C^2 = 1$ . Пусть  $(x_i, y_i, z_i) \in \mathbb{R}^3, i = \overline{1, 3}$  — центры сфер,  $r_i, i = \overline{1, 3}$  — их радиусы. Очевидно, что факт касания сфер плоскости можно записать уравнениями  $Ax_i + By_i + Cz_i + D = \pm r_i, i = \overline{1, 3}$ . Добавим к ним уравнение нормальности

плоскости и получим систему четырех уравнений с четырьмя неизвестными.

### Задача В. Расстояние между отрезками

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

Даны два отрезка в пространстве  $(x_1, y_1, z_1) - (x_2, y_2, z_2)$  и  $(x_3, y_3, z_3) - (x_4, y_4, z_4)$ . Найдите расстояние между отрезками.

#### Формат входного файла

Заданы целые  $x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4$ . Координаты по модулю не превосходят 1000.

#### Формат выходного файла

Выведите искомую величину с точностью не менее 6 знаков после десятичной точки.

#### Пример

b.in	b.out
0 0 0	0.4082482905
1 1 1	
0 0 1	
0 1 0	

### Разбор задачи В. Расстояние между отрезками

Пусть  $p_i \in R^3, i = \overline{1, 4}$ , а  $s_1 = [p_1, p_2], s_2 = [p_3, p_4]$  — отрезки. Рассмотрим параметрическое задание отрезков  $s_1(t) = p_1 + (p_2 - p_1) \cdot t, s_2(t) = p_3 + (p_4 - p_3) \cdot t, t \in [0, 1]$ , а также квадрат евклидова расстояния между точками  $r2(t_1, t_2) = \rho^2(s_1(t_1), s_2(t_2))$ . Это квадратичный полином по переменным  $t_1, t_2$ . Очевидно, минимум составит решение системы

$$\begin{cases} \frac{d}{dt_1} r2(t_1, t_2) = 0 \\ \frac{d}{dt_2} r2(t_1, t_2) = 0. \end{cases} \quad (1)$$

Пусть это  $(t'_1, t'_2)$ . Если  $(t'_1, t'_2) \in [0, 1] \times [0, 1]$ , то ответ  $\sqrt{r2(t'_1, t'_2)}$ , иначе нужно найти расстояния между концами одного отрезка и другим и взять минимум.

## Задача С. Конусы

Имя входного файла:	<code>c.in</code>
Имя выходного файла:	<code>c.out</code>
Ограничение по времени:	4 с
Ограничение по памяти:	256 Мб

Даны  $n$  конусов в пространстве. Никакие два конуса не имеют общих точек. Основания конусов лежат в одной плоскости, и сами конусы лежат по одну сторону от этой плоскости. Можно соединять вершины конусов отрезками, если и только если отрезок не имеет общих точек с конусами.

### Формат входного файла

Первая строка содержит число  $n$  ( $1 \leq n \leq 300$ ). Далее, в  $n$  строках даны по четыре числа  $x_i, y_i, r_i, h_i$  — координаты центра основания конуса, радиус основания, высота конуса.

Все числа целые ( $-500 \leq x_i, y_i \leq 500, 1 \leq r_i \leq 100, 1 \leq h_i \leq 100$ ).

### Формат выходного файла

Выведите единственное целое число — через какое наименьшее количество отрезков можно дойти до  $n$ -го конуса, если стартовать из первого.

### Пример

<code>c.in</code>	<code>c.out</code>
<pre>3 0 0 1 2 0 3 1 3 0 6 1 2</pre>	2

## Разбор задачи С. Конусы

В данной задаче нужно построить граф для видимости вершин конусов друг друга и найти кратчайший по количеству промежуточных ребер путь. Как понять, пересекает ли отрезок конус? Пусть  $s = [p_1, p_2], p_1, p_2 \in \mathbb{R}^3$  — отрезок,  $K$  — конус с высотой  $H$ , радиусом основания  $R$  и центром основания  $c = (x_0, y_0)$ . Без ограничения общности можно считать, что все точки отрезка  $s$  лежат не выше вершины конуса. Заметим, что радиус сечения конуса плоскостью, параллельной  $Oxy$  (только такие сечения и рассматриваются), зависит линейно от высоты, а именно:  $r(h) = R \cdot \frac{H-h}{H}$ . Высота точки отрезка  $s(t)$  линейна по  $t$ , поэтому радиус сечения, лежащего на одной высоте с  $s(t)$ , тоже линеен и равен  $r(s(t).z)$ . Квадрат расстояния

$s(t)$  до оси — квадратичный полином  $\rho(s(t), c)^2$ . Поэтому достаточно решить квадратное неравенство  $\rho(s(t), c)^2 - r^2(s(t)).z \leq 0$  с ограничениями  $0 \leq t \leq 1$ . Для этого можно найти решения уравнения и проверить на принадлежность отрезку  $[0, 1]$  и проверить, что  $0, 1$  являются решениями неравенства.

## Задача D. Снайпер

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

В точке  $S$  находится снайпер. Его цель — убрать врага государства, который едет на велосипеде из точки  $A$  в точку  $B$  по прямой. Пуля летит также по прямой траектории с бесконечной скоростью. На месте действия расположены  $N$  небоскребов в форме параллелепипедов. Траектория пули не может пересекать внутренность зданий. И да, конечно, снайпер стремится сделать смертельный выстрел как можно раньше. Ваша задача — определить координаты врага в момент выстрела.

### Формат входного файла

Первая строка содержит координаты  $S : sx, sy, sz (sz \geq 0)$ , разделенные одним пробелом. Вторая строка содержит координаты точек  $A$  и  $B : ax, ay, bx, by$ , также разделенные пробелом.  $z$ -координата велосипедиста на протяжении всего движения остается равной нулю. Последующие  $N$  ( $0 \leq N \leq 1000$ ) строк содержат числа, разделенные пробелом,  $lx, ly, rx, ry, h$  ( $lx < rx, ly < ry, h > 0$ ) — координаты противоположных концов основания здания и его высоту. Стороны небоскребов параллельны осям декартовой системы координат. Все координаты и высоты — целые и не превосходят по абсолютной величине 100. Гарантируется, что никакие два здания не имеют общих точек, отрезок  $AB$  не пересекается со зданиями,  $S$  не принадлежит никакому параллелепипеду.

### Формат выходного файла

Если врага убрать не удастся, выведите “Impossible”. В противном случае, выведите координаты врага государства в момент убийства с точностью  $10^{-7}$ .

## Пример

<b>d.in</b>	<b>d.out</b>
0 0 2 -4 4 4 4 2 -3 2 -1 3 10 1 -1 4 2 20	-1.3333333333 4.0000000000
0 0 2 4 1 4 -1 1 1 -1 3 1 10	Impossible

## Разбор задачи D. Снайпер

Пусть  $s, a, b \in \mathbb{R}^3$  — положение снайпера, начальная и конечная точки движения велосипедиста. Его движение задается функцией  $c(t) = a + (b - a) \cdot t$ ,  $t \in [0, 1]$ . Назовем параметр  $t'$  успешным, если отрезок  $[s, c(t')]$  не пересекает внутренность никакого параллелепипеда. Очевидно, что если этот отрезок не касается параллелепипедов, то существует успешный параметр  $t'' < t'$ . Поэтому ответ нужно искать среди отрезков, касающихся параллелепипедов. Если отрезок касается границы, то он касается ребра параллелепипеда (в силу ограничений задачи). Поэтому можно перебрать ребра всех параллелепипедов ( $u, v$ ) и найти точки пересечения плоскости, проходящей через  $u, v, s$ , с отрезком  $[a, b]$ . Получим набор параметров-кандидатов  $\{t_1, \dots, t_k\}$ ,  $t_i \in [0, 1]$ ,  $i = \overline{1, k}$ . Не забудем учесть 0. Для каждого проверим, пересекает ли отрезок  $[s, c(t_i)]$  внутренность параллелепипеда. Выберем наименьший.

Как проверить, что отрезок  $[a, b]$  ( $c(t) = a + (b - a) \cdot t$ ) пересекает внутренность параллелепипеда  $A$ ? Так как концы отрезка лежат вне  $A$ , пересечение — это либо интервал  $\subset [a, b]$ , либо пустое множество. Концы интервала соответствуют точкам пересечения отрезка с гранями  $A$ . Пересечем отрезок с гранями, проигнорировав случай, когда отрезок лежит в плоскости грани, и получим набор параметров  $\{t_1, \dots, t_k\}$ ,  $t_i \in (0, 1)$ .  $k = 0$  (множество пусто),  $k = 1, k = 2$ . Других случаев нет. Если  $k = 2$ , проверим, что соответствующие координаты точки лежат строго между наименьшими и наибольшими координатами вершин  $A$ .

## Задача Е. Выпуклая оболочка 3D - 1

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

Даны  $n$  точек в пространстве. Никакие 4 точки не лежат в одной плоскости. Найдите выпуклую оболочку этих точек.

### Формат входного файла

Первая строка содержит число  $t$  — количество тестов. В последующих строках описаны сами тесты. Каждый тест начинается со строки, содержащей  $n$  ( $1 \leq n \leq 1000$ ) — число точек. Далее, в  $n$  строках даны по три числа — координаты точек. Все координаты целые, не превосходят по модулю 500. Общее количество точек не превосходит 22000.

### Формат выходного файла

Для каждого теста выведите следующее. В первую строку выведите количество граней  $t$ . Далее в последующие  $t$  строк выведите описание граней: количество вершин и номера точек в исходном множестве. Точки нумеруются в том порядке, в котором они даны во входном файле. Точки в пределах грани должны быть отсортированы в порядке против часовой стрелки относительно внешней нормали к грани.

### Пример

<b>e.in</b>	<b>e.out</b>
2	4
4	3 0 1 3
0 0 0	3 0 2 1
1 0 0	3 0 3 2
0 1 0	3 1 2 3
0 0 1	6
5	3 0 1 4
0 0 0	3 0 2 1
10 0 0	3 0 4 2
0 10 0	3 1 2 3
10 10 10	3 1 3 4
5 5 10	3 2 4 3

## Разбор задачи Е. Выпуклая оболочка 3D - 1

Данную задачу можно решать методом “разделяй и властвуй”.

Обозначим за  $Plane(p_1, p_2, p_3)$  плоскость с нормалью (внешней нормалью), сонаправленной с векторным произведением  $\overrightarrow{p_1, p_2} \times \overrightarrow{p_1, p_3}$ , где  $p_i \in \mathbb{R}^3, i = \overline{1, 3}$  — точки в пространстве. Иногда будем использовать запись  $Plane(v_1, v_2, v_3)$ ,  $v_i$  — индексы вершин,  $i = \overline{1, 3}$ .

- Если количество точек достаточно мало, например,  $n \leq 7$ , то задачу можно решать перебором всевозможных троек точек и проверкой, что получается грань.
- В противном случае, разделим точки (предварительно отсортируем точки как тройки чисел в лексикографическом порядке) на два примерно равных по мощности множества  $A = \{p_1, \dots, p_k\}$  и  $B = \{p_{k+1}, \dots, p_n\}$  так, что  $|A| - |B| \in [0, 1]$ . Решим задачу построения выпуклой оболочки для множеств  $A$  и  $B$ , получим выпуклые многогранники  $P$  и  $Q$ . Если сумеем объединить многогранники в один за линейное время, то получим алгоритм со сложностью  $O(n \log(n))$ .
- Спроектируем многогранники на плоскость  $Oxy$ . Решим задачу объединения выпуклых многоугольников на плоскости (см. далее, это можно сделать за  $O(n \log(n))$  времени) и получим выпуклый многоугольник  $T$ . После объединения возьмем пару точек, по одной из обоих многоугольников, такую, что они образуют сторону  $T$ . Очевидно, что данная пара точек для исходных многогранников образует ребро  $e = (q_1, q_2)$ ,  $q_1 \in P, q_2 \in Q$ , выпуклой оболочки  $P$  и  $Q$ , а значит,  $A \cup B$ .
- Ребро  $e$  не является вертикальным отрезком, поэтому существует единственная плоскость  $\alpha$ , проходящая через него и параллельная оси  $Oz$ . Все точки лежат по одну сторону от данной плоскости. Пусть  $n_\alpha$  — ее внешняя нормаль. Чтобы найти боковую грань, достаточно выбрать точку  $q$  такую, что нормаль плоскости  $Plane(q, q_1, q_2)$  образует с  $n_\alpha$  наименьший угол. (Здесь используется та же идея, что и в методе “заворачивание подарка”).
- Многогранник удобно хранить как граф (список смежности), при этом все вершины, смежные с данной  $v$ , держать упорядоченными по или против часовой стрелки. Для этого для каждой вершины посмотрим на выходящие из нее ребра. Зафиксируем одно из ребер, его конец  $v_1$ . Подберем к нему ребро  $ee$  такое, что многогранник лежит ниже плоскости  $Plane(v, v_1, v_2)$ , где  $v_2$  — конец ребра  $ee$ . Зафиксируем ребро

с концом  $v_2$ . Проделаем то же самое и т.д. Это не очень эффективно, но так как применяться будет для многогранников с небольшим количеством вершин в листах дерева подзадач, то это не влияет на асимптотику.

- Итак, имеем последовательность вершин  $Ea = (a_1, \dots, a_t)$  многогранника  $P$  и  $Eb = (b_1, \dots, b_s)$  многогранника  $Q$ , последовательность добавленных ребер  $E = ((a_1, b_1), \dots, (a_t, b_s))$ , а также внешнюю нормаль  $n_c$  к последней построенной грани. Чтобы построить следующую грань необходимо перебрать вершины в списке смежности вершины  $a_t$  многогранника  $P$ , аналогично для многогранника  $Q$ , и выбрать такую  $q$ , что нормаль к плоскости  $Plane(a_t, b_s, q)$  образует наименьший угол с  $n_c$ . Если  $q \in P$ , то  $Ea = Ea + q$ ,  $E = E + (q, b_s)$ , иначе  $Eb = Eb + q$ ,  $E = E + (a_t, q)$ . Процедура заканчивается, когда  $a_1 = a_t$  и  $b_1 = b_s$ , при этом из  $Ea, Eb, E$  удаляются последние элементы.
- $Ea$  образует реберную границу многогранника  $P$ , которая отделяет видимую поверхность от невидимой (покрытой новыми боковыми гранями). Аналогично для  $Eb$ . Чтобы удалить из графа невидимые ребра (всю невидимую часть) достаточно обработать  $E$ . Зациклим последовательность  $E$  и будем рассматривать непрерывные отрезки пар вида  $(a, ?)$  с характеристикой  $a$ , где  $?$  — произвольный элемент. Все такие отрезки сгруппируем по характеристике, для каждого отрезка запомнив характеристики предыдущего и следующего отрезков. Этого достаточно, чтобы для каждой вершины на границе разрыва многогранника за линейное время от степени вершины и числа добавляемых ребер удалить невидимые ребра и добавить новые.
- Как объединять выпуклые оболочки на плоскости? Пусть  $(p_0, \dots, p_n)$ ,  $(q_0, \dots, q_k)$  — выпуклые многоугольники, вершины ориентированы против часовой стрелки, последовательные стороны не параллельны. Пусть  $p_c$  — центр масс первого многоугольника. По построению многоугольники могут касаться друг друга, но их внутренности не имеют общих точек. Поэтому  $p_c$  лежит строго вне второго многоугольника. Найдем такие  $i_s, i_f \in \{0, \dots, k\}$ , что  $\overline{p_c, q_{i_s}} \times \overline{q_{i_s}, q_{i_s+1}} > 0$ ,  $\overline{p_c, q_{i_s}} \times \overline{q_{i_s-1}, q_{i_s}} \leq 0$  и  $\overline{p_c, q_{i_f}} \times \overline{q_{i_f}, q_{i_f+1}} \leq 0$ ,  $\overline{p_c, q_{i_f}} \times \overline{q_{i_f-1}, q_{i_f}} > 0$ . Индексы берем по модулю  $k + 1$ .  $q_{i_s}, \dots, q_{i_f}$  — это вершины, которые могут попасть в объединение выпуклых оболочек. Найдем  $j_s, j_f \in \{0, \dots, n\}$  такие, что  $\overline{p_c, q_{i_f}}$  пересекает  $\overline{p_{j_s-1}, p_{j_s}}$  и  $\overline{p_c, q_{i_s}}$  пересекает  $\overline{p_{j_f}, p_{j_f+1}}$ , причем для каждого из двух отрезков с таким свойством верно, что предыдущий таким свойством не обладает. Теперь для концов последовательностей точек  $p_{j_s}, \dots, p_{j_f}$ ,  $q_{i_s}, \dots, q_{i_f}$  нужно проверять то,

что они образуют корректный поворот против часовой стрелки. Если нет, то удаляем, пока это свойство не станет выполняться. Сложность  $O(n + k)$ .

- Результатом построения выпуклой оболочки будет планарный граф. Смежные вершины каждой конкретной вершины отсортированы по полярному углу. Такой граф можно обойти, найдя все его грани (см. алгоритм обхода граней планарного графа в лекции).

## Задача F. Прямые

Имя входного файла:	<code>f.in</code>
Имя выходного файла:	<code>f.out</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася давно интересуется  $n$ -мерным пространством. На уроке математики ему рассказали, что линии в трехмерном пространстве могут совпадать, быть параллельными, пересекаться или скрещиваться. После урока учительница по секрету рассказала, что в  $n$ -мерном пространстве все точно так же. Вася хочет определить, как располагаются прямые, заданные двумя точками.

### Формат входного файла

Для каждого набора входных данных записано целое число  $n$  ( $2 \leq n \leq 50$ ) — размерность пространства. Далее записано  $n$  чисел — координаты точки  $p_1$ , затем  $n$  чисел — координаты точки  $p_2$ , затем —  $r_3$  и  $r_4$ . Все координаты целые, не превосходящие по модулю 2000. Числа разделены пробелами и/или переводами строк.

Входные данные заканчиваются вводом  $n = 0$ . Количество входных наборов не превосходит  $10^4$ .

### Формат выходного файла

Для каждого набора входных данных выведите взаимное расположение прямых. Если прямые пересекаются выведите “cross”, скрещиваются — “skew”, параллельны — “parallel”, совпадают — “equal”. Каждое слово выводите на отдельной строчке.

## Пример

<b>f.in</b>	<b>f.out</b>
3	equal
256 13 117	parallel
516 233 437	skew
9 -196 -187	cross
178 -53 21	
3	
23 167 5	
-133 -223 -671	
230 129 200	
236 144 226	
3	
206 151 224	
263 191 263	
1 134 50	
11 10 273	
3	
96 167 249	
60 122 228	
65 117 232	
60 122 228	
0	

## Разбор задачи F. Прямые

Случай  $R^n$  ничем принципиально не отличается от случая на плоскости или в пространстве. Чтобы узнать, коллинеарны ли вектора  $v_1 = \overrightarrow{p_1, p_2}$  и  $v_2 = \overrightarrow{p_3, p_4}$  найдем их скалярное произведение в квадрате и сравним с произведением квадратов их длин:  $(\sum_{i=1}^n v_{1i} \cdot v_{2i})^2$  и  $\sum_{i=1}^n v_{1i}^2 \cdot \sum_{i=1}^n v_{2i}^2$ . Если равны, значит, прямые либо параллельны, либо совпадают; совпадение отделяется проверкой на коллинеарность векторов  $\overrightarrow{p_1, p_2}$  и  $\overrightarrow{p_1, p_3}$ . В противном случае, расстояние между прямыми единствено и находится аналогично расстоянию между отрезками (см. “Расстояние между отрезками”). Проверка расстояния на нуль разделяет случаи пересечения и скрещивания.

## Задача G. Взаимное расположение прямых

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

Неважно кто N. сидит в точке  $a$  некоторой прямой и умеет ползти со скоростью  $V$ . На другой или первой прямой в точке  $b$  лежит неважно что X., страстно желаемое N.

Помогите N. определить время, которое ему понадобится, чтобы добраться до X. Учтите, что N. в любой момент времени должен оставаться на одной из двух прямых.

### Формат входного файла

Входной файл содержит 5 строк:

- шесть чисел  $x_{11}, y_{11}, z_{11}, x_{12}, y_{12}, z_{12}$  — координаты двух различных точек первой прямой;
- шесть чисел  $x_{21}, y_{21}, z_{21}, x_{22}, y_{22}, z_{22}$  — координаты двух различных точек первой прямой;
- три числа  $a_1, b_1, c_1$  — координаты N;
- три числа  $a_2, b_2, c_2$  — координаты X;
- $V$  — скорость перемещения N.

Все числа целые, не превышающие по модулю  $10^6$ . Гарантируется, что и N., и X. находятся каждый на одной из прямых.

### Формат выходного файла

Минимальное время, необходимое N., чтобы добраться до X. Результат выведите с пятью знаками после десятичной точки. Если N. добраться до X. не сможет, выведите в выходной файл число “−1”.

### Пример

<b>g.in</b>	<b>g.out</b>
0 0 0 5 5 5 6 6 6 9 9 9 0 0 0 10 10 10 1	17.32051

## Разбор задачи G. Взаимное расположение прямых

В данной задаче необходимо найти расстояние между прямыми (см. задачу “Прямые”).

### Задача H. Выпуклая оболочка 3D - 2

Имя входного файла:	<code>h.in</code>
Имя выходного файла:	<code>h.out</code>
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Даны  $n$  точек в пространстве. Никакие 4 точки не лежат в одной плоскости. Найдите выпуклую оболочку этих точек.

#### Формат входного файла

Первая строка содержит число  $t$  — количество тестов. В последующих строках описаны сами тесты. Каждый тест начинается со строки, содержащей  $n$  ( $1 \leq n \leq 1000$ ) — число точек. Далее, в  $n$  строках даны по три числа — координаты точек. Все координаты целые, не превосходят по модулю 500. Общее количество точек не превосходит 2100.

#### Формат выходного файла

Для каждого теста выведите следующее. В первую строку выведите количество граней  $t$ . Далее в последующие  $t$  строк выведите описание граней: количество вершин и номера точек в исходном множестве. Точки нумеруются в том порядке, в котором они даны во входном файле. Точки в пределах грани должны быть отсортированы в порядке против часовой стрелки относительно внешней нормали к грани.

## Пример

<b>h.in</b>	<b>h.out</b>
2	4
4	3 0 1 3
0 0 0	3 0 2 1
1 0 0	3 0 3 2
0 1 0	3 1 2 3
0 0 1	6
5	3 0 1 4
0 0 0	3 0 2 1
10 0 0	3 0 4 2
0 10 0	3 1 2 3
10 10 10	3 1 3 4
5 5 10	3 2 4 3

## Разбор задачи Н. Выпуклая оболочка 3D - 2

Данную задачу можно решать за  $O(n^2)$  методом заворачивания подарка.

- Найдем какую-нибудь грань. Для этого выберем самую нижнюю точку, обозначим  $p_0$ . В качестве второй вершины можно взять точку  $p_1$  такую, что вектор  $\overrightarrow{p_0, p_1}$  образует наибольший угол с вертикальной прямой, проходящей через точку  $p_0$ . Третью точку можно перебирать, проверяя каждый раз, что получилась грань. Все эти операции занимают  $O(n^2)$  времени.
- Суть данного метода состоит в переходе от одной грани к смежным по ребру граням. Выпуклому многограннику можно естественным образом поставить в соответствие планарный граф. По теореме Эйлера этот граф имеет  $O(n)$  ребер и граней. Поэтому, если построение соседних граней выполнять за  $O(n)$  действий, то получим требуемую сложность алгоритма  $O(n^2)$ .

Пусть  $V = (p_1, p_2, p_3)$  — текущая грань, причем вектор  $\overrightarrow{p_1, p_2} \times \overrightarrow{p_1, p_3}$  ( $\times$  — операция векторного произведения) сонаправлен с внешней нормалью к грани, и вершины перечислены в порядке против часовой стрелки, если смотреть с внешней стороны грани. Для определенности зафиксируем ребро  $\overrightarrow{p_1, p_2}$ . Выберем любую точку  $p$ , отличную от  $p_1, p_2, p_3$ , построим плоскость (см. “Выпуклая оболочка 3D-1”) по трем точкам  $p, p_2, p_1$  (порядок таков, чтобы вектор нормали  $\bar{n}$  к плоскости смотрел из многогранника).

Пусть  $\alpha$  — угол между нормалью к  $V$  и  $\bar{n}$ . Очевидно, что  $(p, p_2, p_1)$  является гранью тогда и только тогда, когда  $p$  доставляет минимум угла  $\alpha$ . Аналогично для остальных ребер грани  $V$ .

Можно рассматривать не угол, а скалярное произведение векторов, ортогональных плоскостям, чтобы не терять точность при вычислении.

### Задача I. Выпуклая оболочка 3D - 3

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

Даны  $n$  точек в пространстве. Никакие 4 точки не лежат в одной плоскости. Найдите выпуклую оболочку этих точек.

#### Формат входного файла

Первая строка содержит число  $n$  ( $4 \leq n \leq 100$ ). Далее, в  $n$  строках даны по три числа — координаты точек. Все координаты целые, не превосходят по модулю 500.

#### Формат выходного файла

В первую строку выведите количество граней  $t$ . Далее в последующие  $t$  строк выведите описание граней: количество вершин и номера точек в исходном множестве. Точки нумеруются в том порядке, в котором они даны во входном файле. Точки в пределах грани должны быть отсортированы в порядке против часовой стрелки относительно внешней нормали к грани.

#### Пример

i.in	i.out
4	4
0 0 0	3 0 1 3
1 0 0	3 0 2 1
0 1 0	3 0 3 2
0 0 1	3 1 2 3

### Разбор задачи I. Выпуклая оболочка 3D - 3

Данную задачу можно решать за  $O(n^4)$ . Перебираем тройки точек в качестве вершин граней и проверяем, что остальные точки лежат по одну сторону от плоскости, в которой лежит грань. По тройке точек  $p_1, p_2, p_3$

плоскость аналитически описывается уравнением

$$\det \begin{pmatrix} x - p_1.x & y - p_1.y & z - p_1.z \\ p_2.x - p_1.x & p_2.y - p_1.y & p_2.z - p_1.z \\ p_3.x - p_1.x & p_3.y - p_1.y & p_3.z - p_1.z \end{pmatrix} = 0.$$

Вычисление определителя дает уравнение плоскости в виде  $Ax + By + Cz + D = 0$ . Знак знакового расстояния от точки  $p$  до плоскости

$$\frac{Ap.x + Bp.y + Cp.z + D}{\sqrt{A^2 + B^2 + C^2}}$$

показывает, лежит ли точка ниже, выше или на плоскости. Если расстояние меньше 0, то точка лежит выше плоскости (“потолок” плоскости лежит в направлении вектора  $(A, B, C)$ ).

### Задача J. Расстояние от точки до отрезка

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

Дан отрезок в пространстве  $(x_1, y_1, z_1) - (x_2, y_2, z_2)$  и точка  $(x, y, z)$ . Найдите расстояние от точки до отрезка.

#### Формат входного файла

Заданы целые  $x_1, y_1, z_1, x_2, y_2, z_2, x, y, z$ . Координаты по модулю не превосходят 1000.

#### Формат выходного файла

Выведите искомую величину с точностью не менее 6 знаков после десятичной точки.

#### Пример

j.in	j.out
0 0 0	0.8164965809
1 1 1	
0 0 1	

## Разбор задачи J. Расстояние от точки до отрезка

Пусть  $p_i \in R^3, i = 0, 1, 2$ , а  $s = [p_1, p_2]$  — отрезок. Рассмотрим параметрическое задание отрезка  $s(t) = p_1 + (p_2 - p_1) * t, t \in [0, 1]$ , а также квадрат евклидова расстояния между точками  $r2(t) = \rho^2(s(t), p_0)$ . Это квадратичный полином по  $t$ . Концы графика растут в  $+\infty$ , поэтому можно в качестве точки минимума брать нуль его производной.  $t' = \frac{(p_1 - p_0, p_2 - p_1)}{|p_2 - p_1|^2}$ . Заметим, что параметру  $t'$  на прямой  $\overline{p_1, p_2}$  соответствует точка  $s(t)$  такая, что вектор  $p_2 - p_1$  ортогонален  $s(t) - p_0$ . Таким образом, если  $t' \in [0, 1]$ , то ответ  $\sqrt{r2(t')}$ , иначе  $\min(\rho(p_0, p_1), \rho(p_0, p_2))$ .

## Задача K. Цилиндр

Имя входного файла:	<b>k.in</b>
Имя выходного файла:	<b>k.out</b>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Рассмотрим бесконечный в обе стороны цилиндр, ось которого проходит через центр декартовой системы координат, радиуса  $R$ . В данной задаче вам необходимо вычислить площадь поверхности шара с центром в точке  $c$  радиуса  $r$ , которая содержится в цилиндре.

### Формат входного файла

Первая строка содержит четыре числа  $r, x_c, y_c, z_c$  — радиус шара и координаты его центра. Вторая строка содержит четыре числа  $R, x_v, y_v, z_v$  — радиус цилиндра и координаты точки на его оси (не совпадает с началом координат). Все числа целые, не превосходят по абсолютной величине 1000, радиусы положительны.

### Формат выходного файла

Выведите единственное число — площадь поверхности шара, попавшей во внутренность цилиндра, с точностью не менее 4-х знаков после запятой.

### Пример

<b>k.in</b>	<b>k.out</b>
10 100 0 0 5 1 0 0	168.357443
10 100 0 0 50 0 1 0	0.0000

## Разбор задачи К. Цилиндр

Данную задачу можно решать численным интегрированием. Пусть  $R$  — радиус шара,  $r$  — радиус цилиндра,  $d$  — расстояние между осью цилиндра и центром шара.

Случай  $d = 0$ . Если  $r \geq R$ , то ответ — площадь поверхности шара  $4\pi R^2$ . В противном случае, ответ — результат функции  $\text{calculate}(-r, r)$ .

Случай  $d \neq 0$ . Пусть  $S_1$  — сечение сферы плоскостью, проходящей через центр шара и ортогональной оси цилиндра (очевидно, это окружность). Аналогично  $S_2$  — сечение цилиндра. Если  $d >= R + r$ , то ответ нуль. Если  $r - d \geq R$ , то ответ — площадь сферы. Если  $d + r \leq R$ , то ответ —  $\text{calculate}(d - r, d + r)$ . Оставшийся случай — это случай пересечения  $S_1$  и  $S_2$ . Плоскость, проходящая через две точки пересечения параллельно оси цилиндра, делит сферу внутри цилиндра на две части и находится на расстоянии  $L = (R * R + d * d - r * r) / (2 * d)$  от оси цилиндра. Одна часть — это “крышка” сферы, ее площадь равна  $2\pi R(R - L)$ . Остальная часть считается как  $\text{calculate}(d - r, L)$ .

Функция  $\text{calculate}(a, b)$  численно интегрирует функцию  $f(x)$ , которая выражает длину дуг окружности в сечении сферы плоскостью, параллельной оси цилиндра, на расстоянии  $d + x$  от центра сферы. Можно использовать квадратурную формулу Симпсона.

## Задача L. Разрежем арбуз

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

Вася и Петя решили поделить арбуз на две части (будем считать арбуз идеальным шаром с единичным радиусом). Для этого они делают надрезы. Один надрез представляет собой сектор круга, в плоскости которого осуществляется разрез. Дуга сектора задается двумя точками, не лежащими на одном диаметре шара. Надрезы образуют непрерывную замкнутую ломаную линию на поверхности шара и делят поверхность арбуза на две части, причем для любого отрезка верно, что вся ломаная лежит по одному сторону от плоскости, проходящей через данный отрезок. Считаем, что центр сферы находится в начале координат. Никакие два последовательных звена ломанной не принадлежат одной прямой на сфере.

## Формат входного файла

Первая строка содержит число  $n$  ( $3 \leq n \leq 10$ ) — количество узлов

ломанной. В последующих  $n$  строках описаны точки, заданные тройками целых чисел  $x_i, y_i, z_i, i = \overline{1, n}$  ( $-50 \leq x_i, y_i, z_i \leq 50$ ), точки заданы в произвольном порядке. Точка на сфере получается как пересечение сферы с лучом из  $O$  в точку  $(x_i, y_i, z_i)$ .

### Формат выходного файла

Выведите объем меньшего из двух кусков с точностью не менее 7 знаков после десятичной точки.

### Пример

1.in	1.out
3 1 0 0 0 1 0 0 0 1	0.5235988

### Разбор задачи L. Разрежем арбуз

Найдем пары точек, которые соединены отрезком. Для этого построим плоскость, проходящую через центр шара и две точки и проверим, что остальные точки лежат по одну сторону от плоскости. После этого упорядочим точки так, чтобы соседние точки были соединены отрезком. Рассмотрим тройки последовательных точек  $a, b, c$  и найдем телесный угол между отрезками  $\overrightarrow{a, b}$  и  $\overrightarrow{b, c}$ . Найдем сумму углов  $\alpha$ . Ответом будет  $\alpha - (n - 2) \cdot \pi$ .

## День второй (13.02.2011 г.) КонTEST Жукова Дмитрия Ивановича

### Об авторе...

**Жуков Дмитрий Иванович**, родился в 1986 году. Окончил Орловский Государственный Технический Университет. Неоднократный победитель и призер международных, российских и региональных олимпиад и соревнований по программированию. Разработчик в компании Яндекс.

Основные достижения:

- дипломы 3-ей и 1-ой степени на всероссийской школьной олимпиаде по информатике;
- В 2008 году в составе команды ОрелГТУ занял 13-е место, что является лучшим результатом выступления команд ОрелГТУ на финалах Чемпионата Мира по программированию;
- победитель командной олимпиады по программированию II Международного Молодёжного фестиваля информационных технологий;
- 2-е место в личном зачёте и 3-е командное место в VIII Всероссийской олимпиаде по программированию имени Поттосина 2007 года, победитель индивидуального турнира I Открытой олимпиады ЮФУ по программированию;
- в составе команды ОрелГТУ стал финалистом ACM ICPC – 2007, 7-е место на зимних сборах 2008 года в Петрозаводске, стал победителем Южного четвертьфинала NEERC 2007 года, занял 3-е место на летних сборах 2007 года в Петрозаводске;
- финалист Google Code Jam 2009;
- занял 2-е место на онサイト-раунде VIII Открытого Кубка им. Е.В. Панкратьева по программированию;



- 3-е место и диплом I степени на соревнованиях SnarkNews Winter Series-2011.

## Теоретический материал. Алгоритм Карацубы. Быстрое преобразование Фурье

### 1. Алгоритм Карацубы

Пусть нам необходимо выполнить умножение двух чисел, заданных в системе счисления с основанием  $b$ , длина которых не превосходит  $n$ . Если пользоваться стандартным алгоритмом умножения в столбик, то понадобится  $O(n^2)$  арифметических операций.

Обозначим исходные числа  $A$  и  $B$ . Пусть число  $m$  равно целой части числа  $(n + 1)/2$ . Тогда можно представить  $A$  и  $B$  в виде  $A = A_1 + A_2 \cdot b^m$ ,  $B = B_1 + B_2 \cdot b^m$ . Запишем произведение  $A \cdot B$ , как

$$A \cdot B = (A_1 + A_2 \cdot b^m) \cdot (B = B_1 + B_2 \cdot b^m)$$

$$A \cdot B = A_1 \cdot B_1 + A_2 \cdot B_1 \cdot b^m + A_1 \cdot B_2 \cdot b^m + A_2 \cdot B_2 \cdot b^{2m}$$

$$A \cdot B = A_1 \cdot B_1 + (A_2 \cdot B_1 + A_1 \cdot B_2) \cdot b^m + A_2 \cdot B_2 \cdot b^{2m}$$

Рассмотрим выражение  $(A_1 + A_2) \cdot (B_1 + B_2)$ .

$$(A_1 + A_2) \cdot (B_1 + B_2) = A_1 \cdot B_1 + A_1 \cdot B_2 + A_2 \cdot B_1 + A_2 \cdot B_2$$

$$\text{Обозначим } C = A_1 \cdot B_1, D = A_2 \cdot B_2, E = (A_1 + A_2) \cdot (B_1 + B_2).$$

Тогда получаем:

$$A \cdot B = C + (E - C - D) \cdot b^m + D \cdot b^{2m}$$

Следовательно, для вычисления  $A \cdot B$  потребуется 3 операции умножения чисел вдвое меньшей длины и некоторое количество операций сложения и вычитания, которые выполняются за линейное время.

Общая оценка времени работы алгоритма будет иметь вид  $O(n^{\log_2 3})$ .

Следует заметить, что при небольших значениях  $n$  обычное умножение за  $O(n^2)$  работает быстрее алгоритма Карацубы, так как в нём используются более простые операции. Реализация алгоритма Карацубы обычно представляет собой рекурсивную функцию, в которой выполняется разбиение исходных чисел на части и рекурсивный вызов для этих частей. При этом, если длина перемножаемых чисел меньше некоторого фиксированного значения, то выполняется обычное умножение в столбик.

## 2. Быстрое преобразование Фурье

### Представление многочленов

Обычно многочлены представляют в форме набора коэффициентов. В таком случае операция сложения выполняется за линейное время. С операцией умножения дело обстоит хуже. Простой алгоритм требует  $O(nm)$  операций, где  $n$  и  $m$  — степени многочленов.

Существует и другой способ представления многочленов — задание многочлена его значениями в некоторых точках. Так, если заданы  $n$  различных точек и значения многочлена в этих точках, то найдётся ровно один многочлен, степень которого не превосходит  $n - 1$ , и который принимает в заданных точках эти значения.

Если два многочлена представлены набором пар (точка, значение) и наборы этих точек совпадают, то можно выполнять их сложение, складывая значения в этих точках. Так же можно поступать и при умножении, но следует помнить, что, если степень исходных многочленов была ограничена числом  $n$ , то степень результата будет ограничена числом  $2n$ .

### Быстрое умножение многочленов

Таким образом, общая схема быстрого умножения будет выглядеть так. Сначала вычисляем для каждого многочлена его значение в  $2n$  точках. Затем выполняем поэлементное умножение. После этого находим коэффициенты многочлена, который будет иметь заданные значения в заданных точках.

Если в качестве таких точек выбирать произвольные, тогда для перехода из одного представления многочленов к другому потребуется  $O(n^2)$  операций. Но можно выбрать эти точки таким образом, что переход потребует  $O(n \log n)$  операций.

### Комплексные корни из единицы

Для этого будем считать, что многочлены, которые требуется перемножить — это многочлены от комплексного аргумента. А в качестве таких точек возьмём точки вида  $e^{2\pi ik/n}$  для всех  $k$  от 0 до  $n - 1$ , где  $n$  — некоторая степень двойки. Также можно записать эти точки в виде  $\cos(2\pi k/n) + i \sin(2\pi k/n)$ .

Значение  $w_n = e^{2\pi i/n}$  называется главным значением корня  $n$ -й степени из единицы. Остальные значения являются его степенями. В дальнейшем понадобятся следующие свойства корней из единицы.

- Для любых целых  $n \geq 0$ ,  $k \geq 0$  и  $d > 0$  выполняется  $w_{dn}^{dk} = w_n^k$ .
- Если  $n > 0$  чётное, то квадраты  $n$  комплексных корней  $n$ -й степени из единицы представляют собой  $n/2$  корней  $n/2$ -ой степени из единицы.

- Для любого чётного целого  $n > 0$  выполняется  $w_n^{n/2} = w_2 = -1$ .

### **Дискретное преобразование Фурье**

Мы хотим вычислить многочлен  $A(x) = \sum_{j=0}^{n-1} a_j x^j$  в точках  $w_n^0, w_n^1, \dots, w_n^{n-1}$ . Будем считать  $n$  степенью двойки. Запишем результат в виде  $y_k = A(w_n^k) = \sum_{j=0}^{n-1} a_j w_n^{kj}$ . Вектор  $y = (y_0, y_1, \dots, y_{n-1})$  представляет собой дискретное преобразование Фурье вектора коэффициентов  $a = (a_0, a_1, \dots, a_{n-1})$ . Будем записывать  $y = FFT_n(a)$ .

### **Быстрое преобразование Фурье**

Для вычисления быстрого преобразования Фурье применяется следующий метод. Рассмотрим коэффициенты исходного многочлена  $A$  отдельно с чётными индексами, отдельно — с нечётными. Определим два новых многочлена степени не выше  $n/2$ :

$$\begin{aligned} A^{[0]}(x) &= a_0 + a_2 x + a_4 x^2 + \dots + a_{n-2} x^{n/2-1}, \\ A^{[1]}(x) &= a_1 + a_3 x + a_5 x^2 + \dots + a_{n-1} x^{n/2-1}. \end{aligned}$$

Заметим, что  $A(x) = A^{[0]}(x^2) + x A^{[1]}(x^2)$ .

Тогда после вычисления  $y^{[0]} = FFT_{n/2}(a^{[0]})$  и  $y^{[1]} = FFT_{n/2}(a^{[1]})$  можно вычислить  $y$  следующим образом:

$$\begin{aligned} y_k &= y_k^{[0]} + w_n^k y_k^{[1]}, \\ y_{k+n/2} &= y_k^{[0]} - w_n^k y_k^{[1]}, \end{aligned}$$

для  $k$  от 0 до  $n/2 - 1$ .

Таким образом получаем алгоритм, который вычисляет значения многочлена в  $n$  точках за время  $O(n \log n)$ .

### **Обратное преобразование Фурье**

Обратное преобразование Фурье вычисляется по следующей формуле:

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k w_n^{-kj}.$$

Следовательно, заменив в алгоритме  $w_n^k$  на  $w_n^{-k}$  и поделив результат на  $n$ , получаем алгоритм для вычисления обратного преобразования Фурье.

### **Эффективные реализации FFT**

При вычислении  $FFT$  мы использовали следующие формулы

$$\begin{aligned} y_k &= y_k^{[0]} + w_n^k y_k^{[1]}, \\ y_{k+n/2} &= y_k^{[0]} - w_n^k y_k^{[1]}. \end{aligned}$$

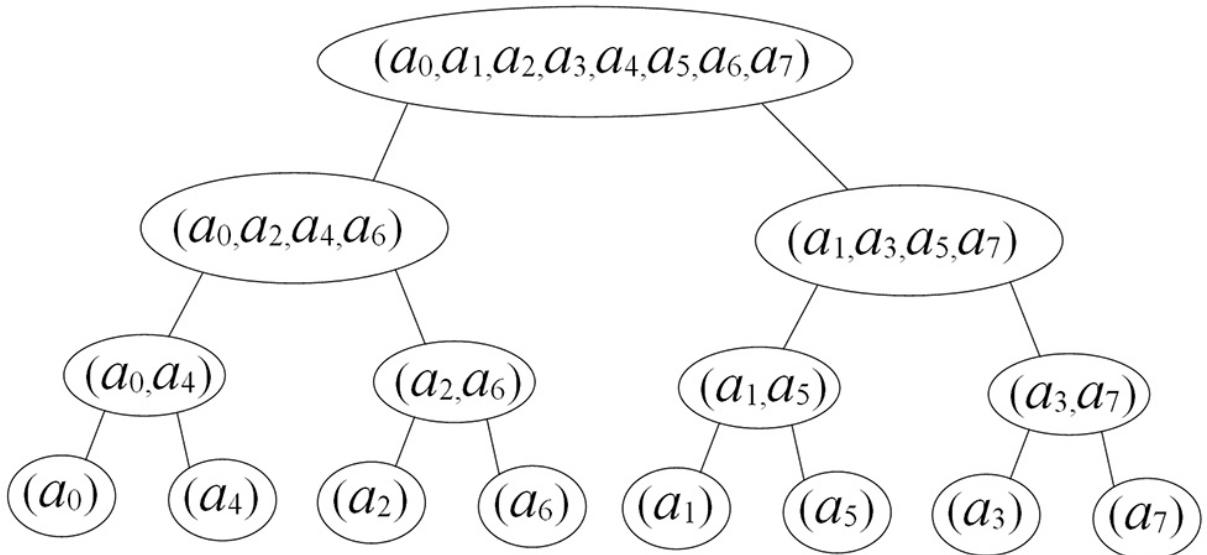
Заметим, что выражение  $w_n^k y_k^{[1]}$  присутствует в обеих формулах. Для того, чтобы не вычислять его два раза, обозначим  $t = w_n^k y_k^{[1]}$ . Тогда получим:

$$y_k = y_k^{[0]} + t,$$

$$y_{k+n/2} = y_k^{[0]} - t.$$

Такую операцию (умножение, сохранение результата в переменной  $t$ , прибавление и вычитание  $t$  из  $y_k^{[0]}$ ) принято называть преобразованием бабочки.

Рассмотрим дерево рекурсивных вызовов функции для вычисления  $FFT$  при  $n = 8$ .



Посмотрим на последовательность индексов элементов, являющихся листьями  $(0, 4, 2, 6, 1, 5, 3, 7)$ .  $k$ -ый элемент этой последовательности определяется следующим образом. Запишем число  $k$  в двоичной системе счисления, дополнив при необходимости лидирующими нулями, чтобы все числа имели одинаковую длину. Переставим биты числа в обратном порядке — полученное число будет  $k$ -ым элементом последовательности.

Также вычисление  $FFT_8$  можно представить следующей схемой:

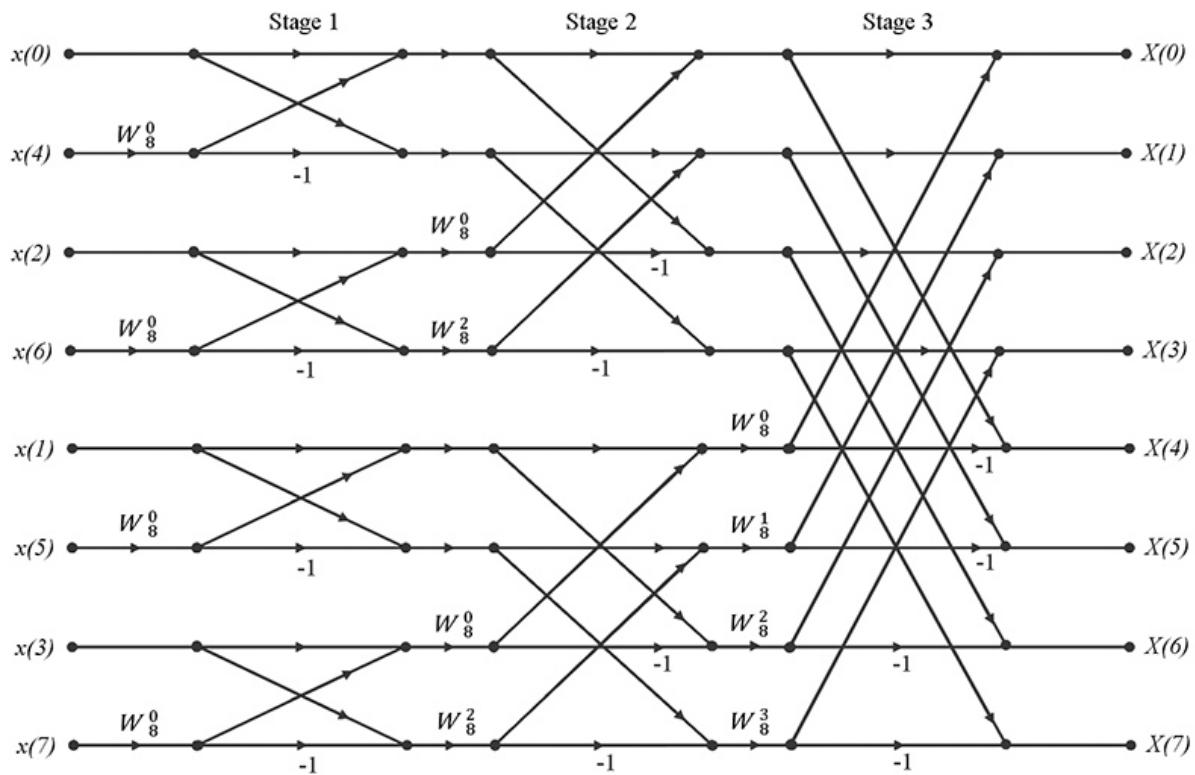


Рисунок 1 – Схема вычисления  $FFT_8$

Такие схемы позволяют вычислять  $FFT$  без использования рекурсии. Но они обладают общим недостатком — алгоритм обращается не к последовательным адресам памяти, в результате чего быстродействие падает.

Можно модифицировать приведённые схемы так, чтобы получить следующий алгоритм:

```
template <typename T>
void step(int k, T *a, T *b, int n)
{
    int h = n / 2;
    int m = n >> k;
    T t, u, v, r = w[0];
    int c = m;
    int d = 0;
    for(int i = 0; i < h; i++)
    {
        t = a[i * 2];
        u = a[i * 2 + 1];
        v = u * r;
        b[i] = t + v;
        b[i + h] = t - v;
        if (!--c)
        {
    }
```

```
    d += c = m;
    r = w[d];
}
}

template <typename T>
void fourier(T *a, T *c, int m)
{
    for (int i = 1; i <= m; i++)
    {
        if (i & 1) step(i, a, c, 1 << m);
        else step(i, c, a, 1 << m);
    }
    if (!(m & 1))
        memcpuy(c, a, (1 << m) * sizeof(c[0]));
}
```

На вход функции `fourier()` подаётся массив элементов, расположенных в порядке, соответствующем обратной перестановке битов их индексов. На выходе получаем элементы в обычном порядке. Количество элементов массива равно  $n = 2^m$ .

В данной реализации удаётся избежать большого количества прыжков по памяти. В связи с этим такая реализация алгоритма даёт существенное сокращение времени вычислений.

## Задачи и разборы

### Задача А. ДНК роботов

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

Последние достижения в технологии синтеза ДНК позволили провести эксперимент по созданию биороботов.

Для облегчения задачи создания ПО для управления роботами было принято решение, что их ДНК будет состоять из  $M = 2^n$  символов для некоторого  $n \geq 2$ . Кроме этого, по техническим причинам это будет не обычная строка, а циклическая, то есть её можно начинать читать с любой позиции.

Одной из целей эксперимента является изучение мутаций биороботов. В результате продолжительных наблюдений было найдено много различ-

ных видов роботов. Для понимания процесса мутации учёным необходимо решить следующую задачу. Для ДНК двух роботов требуется определить коэффициент их похожести. Он вычисляется, как максимальное количество совпадающих символов при наилучшем совмещении этих ДНК. Чем больше символов совпадает, тем лучше совмещение.

Требуется написать программу, которая найдёт наилучшее совмещение двух ДНК.

### **Формат входного файла**

В первой строке входного файла записано одно число  $M$  ( $4 \leq M \leq 131072$ ). В следующих двух строках записаны ДНК двух роботов. Обе ДНК — строки, состоящие ровно из  $M$  символов из множества {‘A’, ‘C’, ‘G’, ‘T’}.

### **Формат выходного файла**

В выходной файл выведите два числа — максимальное количество совпадающих символов и значение оптимального сдвига — неотрицательное количество символов второй ДНК, которые необходимо перенести из конца строки в её начало для достижения наилучшего совмещения.

### **Пример**

<b>a.in</b>	<b>a.out</b>
16	15 1
ACGTACGTACGTACGT CGTACGTACGTACGTC	

### **Разбор задачи А. ДНК роботов**

Так как в строках встречаются только 4 вида символов, решим задачу отдельно для каждого вида. То есть найдём для каждого циклического сдвига количество совпадающих символов ‘A’, затем сделаем то же самое для символов ‘C’, ‘G’ и ‘T’. После этого сложим результаты и найдём номер сдвига, при котором достигается суммарное максимальное количество совпадающих символов.

Рассмотрим решение задачи для одного типа символов (например, символ ‘A’). Возьмём обе строки. Заменим в них все символы ‘A’ на 1, а остальные — на 0. Теперь запишем символы второй строки в порядке  $0, M - 1, M - 2, M - 3, \dots, 2, 1$ . Будем считать символы этих двух строк коэффициентами многочленов  $S1_0 + S1_1x + S1_2x^2 + \dots + S1_{M-1}x^{M-1}$  и  $S2_0 + S2_1x + S2_2x^2 + \dots + S2_{M-1}x^{M-1}$ . Сделаем для этих многочленов

быстрое преобразование Фурье. После этого перемножим значения многочленов в каждой точке. Теперь сделаем обратное преобразование Фурье, не забывая поделить каждое число результата на  $M$ . В полученном массиве  $i$ -ый элемент соответствует количеству совпавших символов для  $i$ -ого циклического сдвига.

Так как у нас 4 вида символов, получается, что в этом решении необходимо выполнить 12 преобразований Фурье — по два прямых и по одному обратному для каждого типа символов. Но количество таких преобразований можно уменьшить. Для этого используются следующие приёмы.

Можно вычислять только одно обратное преобразование вместо четырёх. Так как нас интересует только сумма, можно сложить результаты вычислений для всех символов, а только потом сделать обратное преобразование для полученного массива (складывать многочлены можно как в представлении набором коэффициентов, так и в представлении набором значений).

Второй приём состоит в том, что, когда преобразование Фурье выполняется в поле комплексных чисел, а у нас есть две последовательности из действительных чисел, можно выполнить одно преобразование и после этого за линейное время вычислить результат для обеих последовательностей (см. разбор задачи “Раздвоение”).

Таким образом количество вычислений быстрого преобразования Фурье можно сократить с 12 до 5.

## Задача В. Треугольник

Имя входного файла:	b.in
Имя выходного файла:	b.out
Ограничение по времени:	6 с
Ограничение по памяти:	256 Мб

На плоскости расположено  $N$  ( $3 \leq N$ ) точек. Из них случайным образом выбираются три точки, которые затем соединяются отрезками. Требуется определить математическое ожидание периметра получившегося треугольника, при условии, что каждое множество из трёх точек может быть выбрано с равной вероятностью, а получившийся треугольник может быть вырожденным.

### Формат входного файла

В первой строке входного файла находятся два числа  $H$  и  $W$  ( $1 \leq H, W \leq 700$ ). Далее следует  $H$  строк по  $W$  символов.  $j$ -й символ

$i$ -ой строки равен ‘1’, если есть точка с координатами  $(i, j)$ , иначе в соответствующей позиции стоит символ ‘0’. Гарантируется, что во входных данных представлены как минимум три точки.

## Формат выходного файла

В выходной файл выведите одно число — математическое ожидание периметра получившегося треугольника. Ответ должен отличаться от правильного не более чем на  $10^{-6}$ .

## Примеры

<b>b.in</b>	<b>b.out</b>
11 20 10000000001000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 10000000000000000000	34.142135624
3 3 101 010 101	5.794112550

## Разбор задачи В. Треугольник

Рассмотрим все различные треугольники с вершинами в заданных точках. Легко заметить, что каждый отрезок входит в ответ одинаковое количество раз. Теперь достаточно посчитать среднюю длину отрезка, соединяющего пару исходных точек, умножить её на 3 и вернуть в качестве ответа.

Всего отрезков  $N(N - 1)/2$ . Посчитаем их суммарную длину и поделим на количество. Суммарную длину можно вычислить как  $(\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2})/2$ . Для быстрого вычисления указанной суммы найдём количество отрезков для всех возможных пар разностей

$(x_i - x_j), (y_i - y_j)$ . Для этого пронумеруем точки плоскости следующим образом. Точка  $(0, 0)$  получает номер 0, точка  $(1, 0) — 1$ , и т.д., точка  $(2W - 1, 0) — 2W - 1$ . Далее точка  $(0, 1)$  получает номер  $2W$ , точка  $(1, 1) — 2W + 1$ ,  $(2, 1) — 2W + 2$ , ...,  $(2W - 1, 1) — 4W - 1$ , и т.д.

Рассмотрим последовательность  $A$ ,  $i$ -ый элемент которой равен 1, если в  $i$ -ой точке плоскости записана единица и 0 — в противном случае. Количество отрезков для фиксированной пары  $dx, dy$  ( $|dx| < W, |dy| < H$ ) можно вычислить как  $\sum_{i=0}^{2HW} A_i \cdot A_{i+M}$ , где  $M = dy \cdot 2W + dx$ . А все такие суммы можно быстро вычислить, если взять последовательность  $B$ , элементами которой являются элементы последовательности  $A$ , записанные в обратном порядке. Затем вычислить произведение многочленов, соответствующих этим последовательностям при помощи быстрого преобразования Фурье. После того, как для каждой пары  $(x_i - x_j), (y_i - y_j)$  вычислено количество соответствующих отрезков, умножаем это количество на длину отрезка и прибавляем результат к ответу.

Всего нужных элементов в последовательности  $A$  будет  $2HW$ , следовательно, потребуется вычисление быстрого преобразования Фурье для последовательностей, состоящих из не менее, чем  $4HW$  элементов. Для данной задачи это количество можно оценить, как  $K = 2^{21}$  элементов.

Также можно выполнить два преобразования Фурье вместо трёх, если использовать свойство, описанное в задаче “Раздвоение”.

### Задача С. Монеты

Имя входного файла:	c.in
Имя выходного файла:	c.out
Ограничение по времени:	9 с
Ограничение по памяти:	256 Мб

Двое играют в следующую игру. На столе стоят две стопки монет. Игроки ходят по очереди. Во время хода игрок может взять любое ненулевое количество монет из любой стопки либо взять некоторое одинаковое ненулевое количество монет из обеих стопок сразу. Проигрывает тот, кто не сможет сделать очередной ход.

Дано начальное количество монет в стопках. Требуется определить, кто выиграет при оптимальной игре.

### Формат входного файла

В единственной строке входного файла записаны целые числа  $C_1$  и  $C_2$  ( $1 \leq C_1 \leq 7^7$ ,  $1 \leq C_2 \leq 7^7$ ) — количество монет в стопках.

## Формат выходного файла

В выходной файл выведите `First`, если выигрывает первый игрок, или `Second`, если выигрывает второй.

### Пример

c.in	c.out
1 1	First
1 2	Second
1 3	First

## Разбор задачи С. Монеты

Заметим, что если зафиксировать количество монет в одной из стопок, то найдётся ровно одно целое количество монет для второй стопки, при котором позиция станет проигрышной.

Обозначим количество монет в стопках через  $A$  и  $B$  ( $A \leq B$ ). Пусть  $N = B - A$ . Заметим, что позиция является проигрышной тогда и только тогда, когда  $A = [N \cdot \phi]$ , где  $\phi = \frac{\sqrt{5}+1}{2}$ , а  $[x]$  — целая часть числа  $x$ .

Теперь осталось вычислить число  $\phi$  с достаточной точностью. Например, можно вычислить около  $2 \cdot 10^6$  точных знаков, используя метод последовательных приближений Ньютона.

Возьмём уравнение  $f(x) = 0$ , где  $f(x) = x^2 - 5$ . Число  $\sqrt{5}$  является корнем этого уравнения. В качестве первого приближения можно взять число  $x_0 = 2.24$ . Для вычисления следующих приближений будем использовать формулу

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Подставляем вместо  $f(x_n)$  и её производной соответствующие значения.

$$x_{n+1} = x_n - \frac{x_n^2 - 5}{2x_n}$$

$$x_{n+1} = \frac{2x_n^2 - x_n^2 + 5}{2x_n}$$

$$x_{n+1} = \frac{x_n^2 + 5}{2x_n}$$

При таком способе вычисления количество верных знаков на каждом шаге увеличивается примерно вдвое. Но в полученной формуле используется операция деления на длинное число. Этого можно избежать, если

взять в качестве  $f(x)$  другую функцию. Например,  $f(x) = \frac{1}{x^2} - \frac{1}{5}$ . Получаем:

$$\begin{aligned} x_{n+1} &= x_n - \frac{\frac{1}{x_n^2} - \frac{1}{5}}{-2x_n^{-3}} \\ x_{n+1} &= x_n + \frac{1}{2} \cdot \left( \frac{x_n^3}{x_n^2} - \frac{x_n^3}{5} \right) \\ x_{n+1} &= x_n + \frac{1}{2} \cdot \left( x_n - \frac{x_n^3}{5} \right) \\ x_{n+1} &= \frac{2x_n + x_n - \frac{x_n^3}{5}}{2} \\ x_{n+1} &= \frac{15x_n - x_n^3}{10} \end{aligned}$$

При использовании этой формулы удаётся избежать операции деления на длинное число.

Теперь, зная число  $\sqrt{5}$ , вычисляем число  $\phi$ , умножаем на него разность количества монет в стопках ( $N$ ), берём от результата целую часть и сравниваем с количеством монет в меньшей стопке. В случае равенства выводим **SECOND**, иначе — **FIRST**.

## Задача D. Уравнение

Имя входного файла:	<code>d.in</code>
Имя выходного файла:	<code>d.out</code>
Ограничение по времени:	6 с
Ограничение по памяти:	256 Мб

Дано уравнение вида  $X^N + Y^N \equiv Z^N \pmod{M}$ .

Требуется для фиксированных  $N$  и  $M$  найти количество различных решений этого уравнения. Решением назовём такую тройку натуральных чисел  $(X, Y, Z)$ , что выполняется:

- $1 \leq X \leq Y < M$ ;
- $1 \leq Z < M$ ;
- $X^N + Y^N \equiv Z^N \pmod{M}$ .

## Формат входного файла

В единственной строке входного файла записаны числа  $N$  и  $M$  ( $1 \leq N \leq 7^7$ ,  $1 \leq M \leq 7^7$ ).

## Формат выходного файла

В выходной файл выведите одно число — ответ на задачу.

## Примеры

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

## Разбор задачи D. Уравнение

Обозначим через  $a_i$  количество таких  $X$ , что  $1 \leq X < M$  и  $X^N \mod M = i$ . Переберём все возможные суммы  $(X^N \mod M) + (Y^N \mod M)$  от 0 до  $2M - 2$ . Для фиксированной суммы  $S$  существует  $c_S = \sum_{k=0}^S a_k \cdot a_{S-k}$  способов выбрать пару чисел  $X, Y$  такую, что  $(X^N \mod M) + (Y^N \mod M) = S$ ,  $1 \leq X < M$ ,  $1 \leq Y < M$ .

Просуммируем значения  $c_S$  для всех  $S$  от 0 до  $2M - 2$ . При этом пары различных чисел учитываются дважды, а пары одинаковых — по одному разу. Чтобы всё учитывалось равное количество раз, переберём число  $X$  от 1 до  $M - 1$  и добавим к ответу  $a_{(2X^N \mod M)}$  (количество решений уравнения  $X^N + X^N \equiv Z^N \mod M$ ). Поделив результат на 2, мы получим ответ на задачу.

Для вычисления значений  $c_S$  нужно использовать алгоритм быстрого преобразования Фурье для умножения многочленов.

## Задача E. Разворот битов

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

Пусть  $n = 2^s$ , где  $s \geq 0$  — целое число. Для каждого целого  $i$  от 0 до  $n - 1$  определим значение  $a_i$  следующим образом. Пусть  $b_{s-1} \dots b_1 b_0$  — запись числа  $i$  в двоичной системе счисления (при необходимости дополненная слева нулями до длины  $s$ ). Переставим эти биты в обратном порядке  $(b_0 b_1 \dots b_{s-1})$ . Тогда значением  $a_i$  будет число, двоичной записью которого является  $b_0 b_1 \dots b_{s-1}$ .

Например,  $s = 3, i = 3$ , тогда двоичная запись будет выглядеть, как  $011_2$ , а запись в обратном порядке —  $110_2$ , следовательно, если  $s = 3$ , то  $a_3 = 6$ .

Требуется для заданного числа  $s$  быстро отвечать на запросы на подсчёт суммы  $a_l + a_{l+1} + \dots + a_r$  по модулю  $1000000007$ .

### Формат входного файла

В первой строке входного файла записано целое число  $s$  ( $0 \leq s \leq 31$ ). Во второй строке записано целое число  $k$  — количество запросов ( $1 \leq k \leq 5 \cdot 10^5$ ). В следующих  $k$  строках содержатся сами запросы — пары целых чисел  $l, r$  ( $0 \leq l \leq r < 2^s$ ).

### Формат выходного файла

В выходной файл выведите  $k$  чисел — ответы на соответствующие запросы.

### Примеры

<b>e.in</b>	<b>e.out</b>
3	6
1	
3 3	
2	6
2	2
0 3	
0 1	
25	742367723
1	
17 239	

### Разбор задачи Е. Разворот битов

Для вычисления суммы  $a_l + a_{l+1} + \dots + a_r$  научимся вычислять сумму  $f(k) = a_0 + a_1 + \dots + a_k$ . Тогда ответом будет  $f(r) - f(l - 1)$ .

Для вычисления  $f(k)$  посчитаем, сколько раз в двоичной записи чисел от 0 до  $k$  на  $i$ -ой позиции встретится единица. Обозначим  $k_i$  число  $k$  с обнулёнными битами  $0, 1, \dots, i - 1$ . Если в  $i$ -ом разряде стоит 1, то к количеству следует прибавить  $k - k_i + 1$ . Теперь следует добавить количество для чисел, меньших  $k_i$ . Это будет просто  $k_{i+1}/2$ . Для доказательства рассмотрим все числа от 0 до  $k_{i+1}/2 - 1$ . Вставим в  $i$ -ый разряд нулевой или единичный бит. Таким образом мы получим все числа от 0 до  $k_{i+1} - 1$ . Ровно у половины из них будет стоять единица в  $i$ -ом разряде.

Так как каждый  $i$ -ый бит даёт прибавку к ответу равную  $2^{s-i-1}$ , то домножаем полученное для текущего  $i$  количество на  $2^{s-i-1}$  и прибавляем результат к ответу.

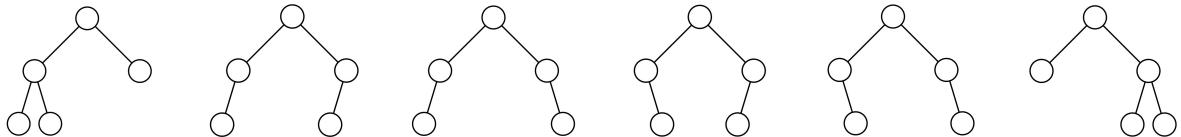
Для каждого запроса необходимо пробежаться по  $s$  битам числа. Получаем решение за  $O(sk)$ , где  $k$  — количество запросов.

### Задача F. АВЛ-деревья

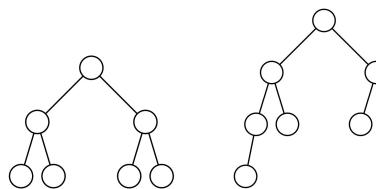
Имя входного файла:	<code>f.in</code>
Имя выходного файла:	<code>f.out</code>
Ограничение по времени:	5 с
Ограничение по памяти:	256 Мб

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1. АВЛ-деревья названы по первым буквам фамилий их изобретателей, Г. М. Адельсона-Вельского и Е. М. Ландиса.

Для фиксированного количества вершин может существовать несколько АВЛ-деревьев. Например, существует шесть АВЛ-деревьев, состоящих из пяти вершин.



Также деревья с одинаковым количеством вершин могут иметь различную высоту. Например, существуют деревья из семи вершин с высотами 2 и 3 соответственно.



Требуется по заданным  $n$  и  $h$  найти количество АВЛ-деревьев, состоящих из  $n$  вершин и имеющих высоту  $h$ . Так как ответ может быть очень большим, требуется найти остаток от деления искомого количества на 786433.

### Формат входного файла

Во входном файле даны числа  $n$  и  $h$  ( $1 \leq n \leq 65535$ ,  $0 \leq h \leq 15$ ).

## Формат выходного файла

Выведите одно число — остаток от деления количества АВЛ-деревьев, состоящих из  $n$  вершин и имеющих высоту  $h$ , на 786433.

### Пример

<b>f.in</b>	<b>f.out</b>
7 3	16

786433 — простое число,  $786433 = 3 \cdot 2^{18} + 1$ .

## Разбор задачи F. АВЛ-деревья

Обозначим через  $D(h, n)$  количество АВЛ-деревьев, состоящих из  $n$  вершин и имеющих высоту  $h$ . Рассмотрим корень этого дерева и его левое и правое поддеревья. Суммарное количество вершин в поддеревьях равно  $n - 1$ , а для их высот возможны 3 варианта:  $h - 1$  для левого и  $h - 1$  для правого,  $h - 2$  для левого и  $h - 1$  для правого,  $h - 1$  для левого и  $h - 2$  для правого.

Допустим, для меньших значений  $h$  ответ посчитан. Тогда значение  $D(h, n)$  можно вычислить по формуле

$$D(h, n) = \sum_{k=0}^{n-1} (D(h-1, k) \cdot D(h-1, n-k-1) + 2D(h-1, k) \cdot D(h-2, n-k-1)).$$

Заметим, что если считать значения  $D(h)$  коэффициентами многочлена  $D(h, 0) + D(h, 1)x + D(h, 2)x^2 + \dots$ , то формулу можно записать в виде  $D(h) = (D(h-1) \cdot D(h-1) + 2D(h-1) \cdot D(h-2))x$ .

Базовый случай — дерево, состоящее из одной вершины, и пустое дерево.  $D(0) = x$ ,  $D(-1) = 1$ .

Для быстрого перемножения многочленов применим быстрое преобразование Фурье. Будем представлять многочлены в виде вектора значений в соответствующих точках. Так как ответ нужно считать по модулю 786433 =  $3 \cdot 2^{18} + 1$ , найдём число  $g$ , степени которого, взятые по модулю 786433, — различные числа от 1 до 786433 − 1. Например, число  $g = 10$ . В качестве корня из единицы можно брать число  $g^{(786433-1)/(2^{16})}$ .

Для умножения двух многочленов будем умножать их значения в соответствующих точках. Для умножения многочлена на  $x$  для каждой точки  $g^k$  (для  $k$  от 0 до 65535) умножим его значение в этой точке на  $g^k$ .

В конце применим обратное преобразование Фурье для вектора  $D(h)$  и выведем значение  $D(h, n)$ .

## Задача G. Многочлен

Имя входного файла: **g.in**  
Имя выходного файла: **g.out**  
Ограничение по времени: 7 с  
Ограничение по памяти: 256 Мб

Многочлен  $P(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$  задан набором коэффициентов  $a_0, a_1, \dots, a_{n-1}$ .

Требуется вычислить значения этого многочлена по модулю  $m$  для всех целых  $x$  от 0 до заданного числа  $k$ .

### Формат входного файла

В первой строке входного файла записаны числа  $n, k$  и  $m$  ( $1 \leq n \leq 2000$ ,  $1 \leq k \leq 200000$ ,  $1 \leq m \leq 10^9$ ). Во второй строке записаны коэффициенты многочлена  $a_0, a_1, \dots, a_{n-1}$  — целые неотрицательные числа, не превосходящие  $10^9$ .

### Формат выходного файла

В выходной файл выведите  $k + 1$  число — остатки от деления значений  $P(0), P(1), \dots, P(k)$  на  $m$ .

### Примеры

<b>g.in</b>	<b>g.out</b>
2 4 239 17 3	17 20 23 26 29
3 5 11 5 8 1	5 3 3 5 9 4

### Разбор задачи G. Многочлен

Для вычисления значений многочлена  $P(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$  в заданной точке можно использовать формулу  $P(x) = (\dots((a_{n-1}x + a_{n-2})x + a_{n-3})x + \dots a_1)x + a_0$ .

Переберём все  $x$  от 0 до  $k$  и вычислим необходимые значения. В результате получаем решение, которое делает  $O(nk)$  операций умножения и взятия остатка от деления на некоторое число  $m$ .

Такое решение не проходит по времени. Его можно ускорить, уменьшив количество медленных операций.

Вычислим значения  $P(0), P(1), \dots, P(n - 1)$ . Для этого необходимо  $O(n^2)$  операций mod. После этого значения  $P(n), P(n+1), \dots, P(k)$  можно вычислять, используя только операции сложения и вычитания.

Обозначим  $A(0, 0) = P(0)$ ,  $A(0, 1) = P(1), \dots, A(0, n) = P(n), \dots$ . Для  $i$  от 1 до  $n - 1$  и  $j \geq 0$  будем считать, что  $A(i, j) = A(i - 1, j + 1) - A(i - 1, j) \bmod m$ . Заметим, что при вычитании числа в диапазоне от 0 до  $m - 1$  операцию  $\bmod$  следует заменить на проверку результата на неотрицательность и при необходимости прибавить к ответу  $m$ . При сложении следует поступать подобным образом, сравнивая результат с  $m$  и производя при необходимости вычитание из результата числа  $m$ .

Рассмотрим последовательности чисел  $A(i, 0)$ ,  $A(i, 1)$ ,  $A(i, 2) \dots$  и  $A(i + 1, 0)$ ,  $A(i + 1, 1)$ ,  $A(i + 1, 2) \dots$ . Заметим, что, если первая последовательность представляет собой значения многочлена степени  $q$ , то вторая последовательность представляет собой значения многочлена степени  $q - 1$ . Доказательство следует из того, что  $(x + 1)^q - x^q = (\sum_{j=0}^q C_q^j x^j) - x^q = \sum_{j=0}^{q-1} C_q^j x^j$ .

Следовательно, все значения  $A(n - 1, 0)$ ,  $A(n - 1, 1)$ ,  $A(n - 1, 2), \dots$  равны между собой. А зная значения  $A(i, j)$  и  $A(i - 1, j)$ , можно вычислить значение  $A(i - 1, j + 1)$ , как сумму первых двух. Таким образом, для вычисления очередного значения  $A(0, j)$  потребуется  $O(n)$  операций сложения и вычитания.

В итоге получаем решение, которое использует  $O(n^2)$  операций умножения и взятия остатка от деления и  $O(nk)$  операций сложения и вычитания.

## Задача Н. Раздвоение

Имя входного файла:	<code>h.in</code>
Имя выходного файла:	<code>h.out</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Обозначим две последовательности действительных чисел  $x(k)$  и  $y(k)$ . Определим последовательность комплексных чисел  $z(k)$ :  $z(k) = x(k) + iy(k)$ .

Пусть  $FFT_N(k, z) = \sum_{n=0}^{N-1} z_n e^{2\pi i kn/N}$ . Аналогичным образом определяются  $FFT_N(k, x)$  и  $FFT_N(k, y)$ .

Требуется по вычисленным значениям  $FFT_N(k, z)$  восстановить значения  $FFT_N(k, x)$  и  $FFT_N(k, y)$ .

### Формат входного файла

В первой строке входного файла записано целое число  $N$  ( $1 \leq N \leq 2^{30}$ ,  $N$  является степенью двойки). Далее следуют целые неотрицательные чис-

ла  $A, B, C, D, E, F$ , не превосходящие 1000. Для экономии времени ввода значения  $FFT_N(k, z)$  нужно будет вычислять по следующим формулам:

$$FFT_N(k, z).real = ((A + B \cdot k) \text{ xor } (C \cdot k)) \cdot 10^{-3},$$

$$FFT_N(k, z).imag = ((D + E \cdot k) \text{ xor } (F \cdot k)) \cdot 10^{-3},$$

где  $FFT_N(k, z).real$  и  $FFT_N(k, z).imag$  — действительная и мнимая части соответственно.

Затем дано число  $M$  — количество запросов ( $1 \leq M \leq 10^5$ ). Далее следуют  $M$  целых чисел  $q_j$  ( $0 \leq q_j < N$ ).

### Формат выходного файла

В выходной файл выведите  $M$  строк. В  $j$ -ой строке — значения  $FFT_N(q_j, x)$  и  $FFT_N(q_j, y)$ . Значения должны отличаться от правильных не более, чем на  $10^{-4}$ .

### Примеры

<b>h.in</b>	<b>h.out</b>
2	1.0 0.0 0.0 0.0
1000 0 0 0 0 0	1.0 0.0 0.0 0.0
2	
0 1	
4	0.000 0.000 0.500
0 100 300 500 100 200	0.000
4	0.504 0.140 0.516
0 1 2 3	0.176 0.656 0.000 0.812 0.000 0.504 -0.140 0.516 -0.176
1048576	540.737 -1587.741
999 998 997 996 995	1589.778 539.689
994	2404.809 531.421
3	1359.578 1569.751
17 239239 2011	3678.277 -523.243 526.382 3664.887

### Разбор задачи Н. Раздвоение

Будем обозначать  $a^*$  — сопряжённое к числу  $a$ . Из  $z(n) = x(n) + iy(n)$  следует, что

$$x(n) = (z(n) + z(n)^*)/2$$

$$y(n) = -i(z(n) - z(n)^*)/2$$

Следовательно,

$$FFT_N(k, x) = (FFT_N(k, z) + FFT_N(k, z^*))/2$$

$$FFT_N(k, y) = -i(FFT_N(k, z) - FFT_N(k, z^*))/2$$

Для вычисления  $FFT_N(k, z^*)$  можно воспользоваться следующим свойством:

$$\begin{aligned} FFT_N(k, z^*) &= \sum_{n=0}^{N-1} z(n)^* e^{2\pi i kn/N} = \left( \sum_{n=0}^{N-1} z(n) e^{-2\pi i kn/N} \right)^* \\ &= \left( \sum_{n=0}^{N-1} z(n) e^{2\pi i (N-k)n/N} \right)^* \end{aligned}$$

Значит,  $FFT_N(k, z^*) = FFT_N(N - k, z)^*$ .

Получаем следующие формулы для вычисления:

$$FFT_N(k, x) = (FFT_N(k, z) + FFT_N(N - k, z^*))/2$$

$$FFT_N(k, y) = -i(FFT_N(k, z) - FFT_N(N - k, z^*))/2$$

Следовательно, для вычисления  $x(n)$  и  $y(n)$  необходимы значения  $FFT_N(k, z)$  и  $FFT_N(N - k, z)$ . Их следует вычислять по формуле, которая дана в условии. Осталось не забыть про случай  $k = 0$ . В этом случае нужно только значение  $FFT_N(0, z)$ .

## Задача I. Уравнение 2

Имя входного файла: `i.in`

Имя выходного файла: `i.out`

Ограничение по времени: 4 с

Ограничение по памяти: 256 Мб

Дано уравнение вида  $X^N + Y^N \equiv Z^N \pmod{M}$ .

Требуется для фиксированных  $N$  и  $M$  найти количество различных решений этого уравнения. Решением назовём такую тройку натуральных чисел  $(X, Y, Z)$ , что выполняется:

- $1 \leq X \leq Y < M$ ;
- $1 \leq Z < M$ ;
- $X^N + Y^N \equiv Z^N \pmod{M}$ .

## Формат входного файла

В единственной строке входного файла записаны числа  $N$  и  $M$  ( $1 \leq N \leq 50000$ ,  $1 \leq M \leq 50000$ ).

## Формат выходного файла

В выходной файл выведите одно число — ответ на задачу.

## Примеры

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

## Разбор задачи I. Уравнение 2

Обозначим через  $a_i$  количество таких  $X$ , что  $1 \leq X < M$  и  $X^N \bmod M = i$ . Переберём все возможные суммы  $(X^N \bmod M) + (Y^N \bmod M)$  от 0 до  $2M - 2$ . Для фиксированной суммы  $S$  существует  $c_S = \sum_{k=0}^S a_k \cdot a_{S-k}$  способов выбрать пару чисел  $X, Y$  такую, что  $(X^N \bmod M) + (Y^N \bmod M) = S$ ,  $1 \leq X < M$ ,  $1 \leq Y < M$ .

Просуммируем значения  $c_S$  для всех  $S$  от 0 до  $2M - 2$ . При этом пары различных чисел учитываются дважды, а пары одинаковых — по одному разу. Чтобы всё учитывалось равное количество раз, переберём число  $X$  от 1 до  $M - 1$  и добавим к ответу  $a_{(2X^N \bmod M)}$  (количество решений уравнения  $X^N + X^N \equiv Z^N \bmod M$ ). Поделив результат на 2, мы получим ответ на задачу.

Для вычисления значений  $c_S$  нужно использовать алгоритм быстрого преобразования Фурье либо алгоритм Карацубы для умножения многочленов.

## Задача J. Разворот битов 2

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

Пусть  $n = 2^s$ , где  $s \geq 0$  — целое число. Для каждого целого  $i$  от 0 до  $n - 1$  определим значение  $a_i$  следующим образом. Пусть  $b_{s-1} \dots b_1 b_0$  — запись числа  $i$  в двоичной системе счисления (при необходимости дополненная слева нулями до длины  $s$ ). Переставим эти биты в обратном порядке  $(b_0 b_1 \dots b_{s-1})$ . Тогда значением  $a_i$  будет число, двоичной записью которого является  $b_0 b_1 \dots b_{s-1}$ .

Например,  $s = 3, i = 3$ , тогда двоичная запись будет выглядеть, как  $011_2$ , а запись в обратном порядке —  $110_2$ , следовательно, если  $s = 3$ , то  $a_3 = 6$ .

Требуется для заданного числа  $s$  быстро отвечать на запросы на подсчёт суммы  $a_l + a_{l+1} + \dots + a_r$  по модулю  $1000000007$ .

### Формат входного файла

В первой строке входного файла записано целое число  $s$  ( $0 \leq s \leq 25$ ). Во второй строке записано целое число  $k$  — количество запросов ( $1 \leq k \leq 5 \cdot 10^5$ ). В следующих  $k$  строках содержатся сами запросы — пары целых чисел  $l, r$  ( $0 \leq l \leq r < 2^s$ ).

### Формат выходного файла

В выходной файл выведите  $k$  чисел — ответы на соответствующие запросы.

### Примеры

j.in	j.out
3	6
1	
3 3	
2	6
2	2
0 3	
0 1	
25	742367723
1	
17 239	

### Разбор задачи J. Разворот битов 2

См. разбор задачи “Разворот битов”.

Также в этой задаче ограничения позволяют сгенерировать массив  $a$ . После этого сделаем массив  $a$  массивом частичных сумм. Для этого пройдем по массиву в порядке возрастания индексов и заменим каждый элемент на его сумму со значением предыдущего элемента. Ответ на запрос — это просто разность двух частичных сумм.

## Задача К. Дуэль

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

Двое дуэлянтов решили выбрать в качестве места проведения поединка тёплую аллею. Вдоль этой аллеи растёт  $n$  деревьев и кустов. Расстояние между соседними объектами равно одному метру. Дуэль решили проводить по следующим правилам. Некоторое дерево выбирается в качестве стартовой точки. Затем два дерева, находящихся на одинаковом расстоянии от исходного, отмечаются как места для стрельбы. Дуэлянты начинают движение от стартовой точки в противоположных направлениях. Когда соперники достигают отмеченных деревьев, они разворачиваются и начинают стрелять друг в друга.

Дана схема расположения деревьев вдоль аллеи. Требуется определить количество способов выбрать стартовую точку и места для стрельбы согласно правилам дуэли.

### Формат входного файла

Во входном файле содержится одна строка, состоящая из символов ‘0’ и ‘1’ — схема аллеи. Деревья обозначаются символом ‘1’, кусты — символом ‘0’. Длина строки не превосходит 100000 символов.

### Формат выходного файла

Выведите количество способов выбрать стартовую точку и места для стрельбы согласно правилам дуэли.

### Примеры

<b>k.in</b>	<b>k.out</b>
101010101	4
101001	0

В первом примере возможны следующие конфигурации дуэли (стартовое дерево и деревья для стрельбы выделены жирным шрифтом): **101010101**, **101010101**, **101010101** и **101010101**.

### Разбор задачи К. Дуэль

Рассмотрим нашу строку  $S$  из нулей и единиц длиной  $N$ . Пронумеруем её символы слева направо последовательно целыми неотрицательными

числами  $(0, 1, \dots, N - 1)$ . Пусть  $a_i = 1$ , если  $i$ -ый символ строки  $S$  равен единице, и  $a_i = 0$  — в противном случае.

Тогда переберём в качестве стартового дерева все позиции от 0 до  $N - 1$ . Пусть очередная позиция имеет номер  $i$  и  $a_i = 1$ . Тогда количество способов выбрать остальные две позиции для стрельбы равно

$$c_i = \sum_{k=0}^{i-1} a_k \cdot a_{2i-k} = ((\sum_{k=0}^{2i} a_k \cdot a_{2i-k}) - 1)/2.$$

Будем считать значения  $a_i$  коэффициентами многочлена  $a_0 + a_1x + a_2x^2 + \dots + a_{N-1}x^{N-1}$ . Тогда возьмём многочлен  $b = a^2$ .

Заметим, что  $b_{2i} = \sum_{k=0}^{2i} a_k \cdot a_{2i-k}$ .

Следовательно, для вычисления количества способов выбрать остальные две позиции для стрельбы при фиксированной центральной достаточно вычислить коэффициенты многочлена  $b$ . Тогда  $c_i = (b_{2i} - 1)/2$ .

Ответ — это сумма  $c_i$  для всех  $i$ , при которых  $a_i = 1$ .

Так как длина строки ограничена числом 100000, для умножения многочленов можно использовать алгоритм Карацубы или быстрое преобразование Фурье.

## Задача L. Произведение

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

Требуется найти произведение двух целых чисел.

### Формат входного файла

В каждой из двух строк входного файла записано целое число, состоящее не более чем из 239000 цифр.

### Формат выходного файла

В выходной файл выведите одно целое число — произведение этих чисел.

## Примеры

1.in	1.out
2	4
2	
-1	-1
1	

## Разбор задачи L. Произведение

Допустим, хотя бы один из множителей равен нулю. Тогда результатом умножения будет число 0. Иначе сначала узнаем знак произведения. Если ровно один множитель меньше нуля, а другой — больше, тогда результат будет меньше нуля.

Теперь осталось вычислить произведение двух натуральных чисел. Для этого можно использовать алгоритм Карацубы или быстрое преобразование Фурье. Затем нужно вывести результат, при необходимости поставив перед ним знак ‘-’.

## Задача M. Сфера

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

Требуется расположить на поверхности сферы  $N$  точек так, чтобы минимальное из попарных расстояний между этими точками было максимально возможным.

## Формат входного файла

В единственной строке входного файла записаны целые числа  $R$  ( $1 \leq R \leq 1000$ ) и  $N$  ( $2 \leq N \leq 4$ ) — радиус сферы и количество точек соответственно.

## Формат выходного файла

В выходной файл выведите одно действительное число — максимально возможное значение минимального из попарных расстояний между этими точками.

Выводить следует первые 10000 символов ответа (включая разделитель целой и дробной частей) без округления.

## Примеры

<b>m.in</b>	<b>m.out</b>
1 2	2.00000...000
11 3	19.0525...344

## Примечание

Для экономии бумаги 9990 символов ответа в примерах заменены на ‘...’.

## Разбор задачи М. Сфера

См. разбор задачи “монеты” (вычисление числа  $\sqrt{5}$ ).

В данной задаче потребуется вычисление чисел  $\sqrt{3}$  и  $\sqrt{6}$ .

Для  $N = 2$  ответ на задачу  $2R$ . Просто умножаем, выводим и дополняем разделителем целой и дробной частей и необходимым количеством нулей.

Для  $N = 3$  ответ на задачу  $R\sqrt{3}$  (правильный треугольник с вершинами на сфере).

Для  $N = 4$  ответ на задачу  $\frac{2}{3}R\sqrt{6}$  (правильный тетраэдр с вершинами на сфере).

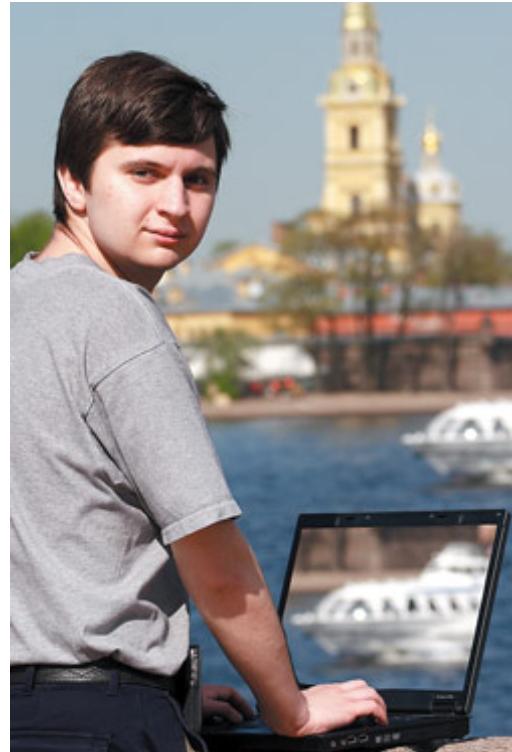
## День третий (14.02.2011 г.) Конкурс Дворкина Михаила Эдуардовича

### Об авторе...

**Дворкин Михаил Эдуардович**, исследователь в Лаборатории вычислительной биологии Академического университета РАН, преподаватель в лицее “Физико-техническая школа”, аспирант СПбГУ ИТМО. Раньше много участвовал в олимпиадах по программированию, сейчас же больше занимается их организацией, а накопленный опыт передает подрастающему поколению. На ТопКодере и в Живом журнале использует псевдоним Дарнли.

Основные достижения:

- Золотой медалист Чемпионата мира по программированию ACM ICPC 2007 года.



### Теоретический материал. Динамическое программирование по профилю и по изломанному профилю

Часто метод динамического программирования излагается на примере задачи о замощении поля “доминошками”  $2 \times 1$ . Чтобы не повторять сложившиеся штампы, рассмотрим близкую, но все же несовпадающую задачу.

Сколькими способами можно замостить прямоугольник  $h \times w$  вертикальными “триминошками” ( $3 \times 1$ ) и горизонтальными “доминошками” ( $1 \times 2$ )?

Под замощением подразумевается укладка данных фигур так, чтобы они не выходили за границу прямоугольника, и каждая клетка прямоугольника была покрыта ровно одной фигуруй.

Эта формулировка — устоявшаяся. Введем альтернативную формулировку той же задачи (зачем — будет объяснено далее): сколькими способами можно заполнить прямоугольник  $h \times w$  буквами “Y”, “E”, “S”, “N” и “O”, так чтобы все буквы “Y”, “E” и “S” шли друг за другом по вертикали именно

в таком порядке (то есть соответствовали “тrimиношкам”), а буквы “N” и “O” шли по горизонтали именно в таком порядке (то есть соответствовали “доминошкам”)? Ясно, что суть задачи не поменялась нисколько.

Пример замощения прямоугольника  $5 \times 4$ :

NONO  
YNOY  
ENOE  
SNOS  
NONO

В такой формулировке гораздо удобнее говорить о частично заполненных прямоугольниках. Например, в вышеприведенном примере левый столбец в некотором смысле “заполнен” — с той лишь оговоркой, что к его правой границе примыкают две буквы “N”, которые должны быть дополнены буквами “O” в следующем столбце.

И в общем случае, если рассмотреть несколько первых столбцов какого-либо корректно замощенного прямоугольника, они представляют собой заполненный прямоугольник, в котором, “неидеальны” только (возможно) стоящие вплотную к правой границе буквы “N”. Таковые могут присутствовать в любой строке.

А теперь сделаем ключевое замечание: для “ дальнейшего” заполнения прямоугольника справа от этих столбцов важна только эта информация: *в каких именно строках к правой границе прилегает буква “N”*.

Эту информацию будем называть *профилем*. Он содержит  $h$  бит — по одному на каждую строчку. Соответственно, его можно закодировать числом от 0 до  $2^h - 1$ . Если записать его в двоичной записи, единицы будут означать, например, присутствие в соответствующей позиции буквы “N”, нули — ее отсутствие.

Итак, все готово для применения метода динамического программирования:

Обозначим  $a_{ij}$  число способов заполнить первые  $i$  столбцов буквами “Y”, “E”, “S”, “N” и “O” так, чтобы множество строк, в которых справа оказалась буква “N” составляло профиль  $j$  (а во всем остальном эти столбцы были заполнены правильно).

Пример замощения 3 столбцов с профилем  $10001_2$ :

NON  
YNO  
ENO  
SNO  
NON

В частности, ответ на задачу — значение  $a_{w0}$ , то есть количество способов заполнить  $w$  столбцов абсолютно корректно, то есть без каких бы то ни было букв “N” справа; это и кодируется профилем 0.

Отметим, что размер этой матрицы (т. е. таблицы) —  $w \times 2^h$ , линейный по одному измерению, экспоненциальный по другому. Это неравноправие является отличительной и, видимо, неисправляемой особенностью метода динамического программирования по профилю.

Что же происходит с нулевым слоем ( $a_{0*}$ )? Единственный способ заполнить 0 столбцов — не поставить ничего; у этой расстановки профиль 0. Значит,  $a_{00} = 1$ , и это единственное ненулевое значение в нулевом слое.

Пусть известны все значения  $i$ -го слоя. То есть для всякого профиля  $j$  известно  $a_{ij}$  — количество способов заполнить  $i$  столбцов, получив справа профиль  $j$ .

Вычислим значения  $(i+1)$ -го слоя. Сколько существует способов получить профиль  $k$ ? Для этого надо перейти к нему от какого-то профиля  $j$ , который имел место, когда были заполнены  $i$  столбцов. Таким образом:

$$a_{i+1,k} = \sum_{j=0}^{2^h-1} w_{jk} a_{ij}$$

Здесь  $w_{jk}$  — число способов перейти от слоя  $j$  к слою  $k$  (число замощений очередного столбца, обеспечивающих такой переход). Если эта матрица (т. е. таблица) будет известна, то по приведенной выше формуле можно вычислить все значения 1-го слоя, затем 2-го и так далее до искомого  $w$ -го.

Что ж, в каком случае существует столбец, такой что без него прямоугольник имеет профиль  $j$ , а с ним — профиль  $k$ ? Рассмотрим все (горизонтальные) домино, которые имеют с этим столбцом общую клетку. Те, что выпирают влево, в точности соответствуют единицам в числе  $j$ , ведь у них слева — буква “N”. Те, что выпирают вправо, в точности соответствуют единицам в числе  $k$ .

Переход от профиля  $100001_2$  к профилю  $010000_2$ :

NO.

. NO

. \* .

. \* .

. \* .

NO.

В частности, если у чисел  $j$  и  $k$  есть хотя бы одна единица в одной и той же позиции (т. е. побитовое И этих двух чисел не равно нулю), то от профиля  $j$  к профилю  $k$  (за один столбец) перейти нельзя.

Если же общих единиц у них нет, то следует убедиться, что оставшиеся (не занятые горизонтальными домино) клетки можно покрыть вертикальными тримино. А сделать это можно тогда и только тогда, когда все группы из подряд идущих незанятых клеток имеют длину, делящуюся на три. Причем в этом случае каждая такая группа заполняется единственным образом.

Итак,  $w_{jk} = 1$ , если  $(j \text{ and } k) = 0$ , и в числе  $(j \text{ or } k)$  все группы из подряд идущих нулей (включая старшие нули, которые надо учесть с особой аккуратностью) имеют длину кратную трем. Иначе  $w_{jk} = 0$ .

Зная матрицу  $w$  (или даже не храня ее, а вычисляя ее значения заново всякий раз; на асимптотику это не влияет), вычисление каждого слоя занимает время  $O(2^h \cdot 2^h) = O(4^h)$ . И общее время работы, необходимое для вычисления  $a_{w0}$  составляет  $O(4^h w)$ .

### **Динамическое программирование по изломанному профилю**

Рассмотрим прием, позволяющий уменьшить основание экспоненты в асимптотике времени работы. Будем рассматривать профиль не после заполнения целиком некоторого количества столбцов, а после заполнения некоторого количества столбцов, а также верхней части очередного столбца.

Изломанный профиль 10010<sub>2</sub>:

NON*	NON*
YNO	NOY
ENO	YYE
SN*	EE*
NO	SS

Динамическое программирование будет применено следующим образом: обозначим  $a_{ixj}$  количество способов заполнить целиком  $i$  столбцов и  $x$  клеток ( $i + 1$ )-го столбца, так чтобы получился профиль  $j$ .

Что такое профиль в случае изломанной границы заполненной области? Рассмотрим по одной клетке в каждой строке — самой левой незаполненной. У нее могут быть два “состояния”: 1) она должна “закрывать” какую-то из уже выставленных клеток (слева или, на этот раз, сверху от нее); 2) она не должна ничего “закрывать”.

При этом, кого именно она должна “закрывать” не имеет значения. В приведенном выше примере один и тот же профиль получен двумя спосо-

бами, причем нижняя “закрывающая” клетка (они отмечены звездочками) в одном случае закрывает домино, в другом — тримино.

Итак, профиль снова занимает  $h$  бит — один бит на каждую строку.

Рассмотрим шаг динамического программирования. Пусть существует  $a_{ixj}$  способов получить профиль  $j$ , когда заполнены  $i$  столбцов целиком и  $x$  клеток следующего столбца.

Переход к новому состоянию динамического программирования будет состоять в выставлении одной (!) буквы в очередную  $(x + 1)$ -ю клетку обрабатываемого  $(i + 1)$ -го столбца. К счастью, вариантов  $O(1)$ .

Рассмотрим  $(x + 1)$ -ю цифру двоичной записи профиля  $j$ . Она говорит о том, должна или нет новая буква закрывать что-либо уже выставленное ранее.

Если да, то вариант перехода ровно один — выставить именно то, что требуется. При этом  $i$  остается прежним,  $x$  увеличивается на 1, а  $(x + 1)$ -я цифра числа  $j$  становится нулем (подумайте, почему это так).

Если же новая буква не должна ничего закрывать, то это может быть только буква “Y” или “N”, означающая соответственно начало нового тримино или домино.

Новое домино можно начать всегда, а вот тримино — не всегда. Для этого в двух следующих (снизу) позициях в профиле должны быть нули. Действительно, если в  $(x + 2)$ -й или  $(x + 3)$ -й позиции в профиле стоит единица, то в соответствующую строку необходимо поставить, по всей видимости, букву “O”, закрывающую соответствующую букву “N” — и букву “E” или “S” поставить уже не получится. (А кроме того, указанные две цифры должны для начала существовать: если речь идет о нижней клетке столбца или о второй снизу, то начинать здесь тримино невозможно).

Если все условия соблюdenы и тримино начать в  $(x + 1)$ -й клетке можно, то новое состояние динамического программирования будет таково:  $i$  прежнее,  $x$  увеличивается на 3, в профиле цифры с  $(x + 1)$ -й по  $(x + 3)$ -ю становятся нулями (поймите, отчего это так).

Во всех случаях, если  $x$  стал равен  $h$ , это соответствует окончанию работы со столбцом — тогда  $x$  обнуляется, а  $i$  увеличивается на единицу.

Искомой величиной при данном подходе является  $a_{w00}$  — число способов целиком заполнить  $w$  столбцов без незавершенностей справа.

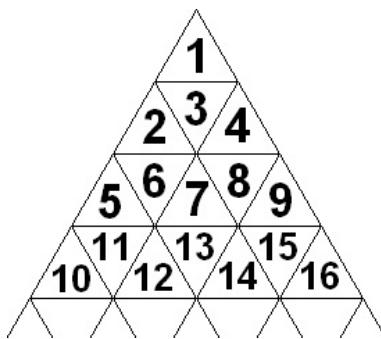
Время работы алгоритма составляет  $O(w \cdot h \cdot 2^h)$ .

## Задачи и разборы

### Задача А. Треугольный король

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

После того, как лютпен-пролетарские пешки устроили бунт на шахматном поле, белый король был вынужден искать политического убежища в другой стране. После долгих скитаний он поселился в стране, хоть как-то напоминающей ему его родину. Вот как выглядит карта этой страны:



Здесь он стал скитаться по полю, переходя с одной клетки на другую, но только если у них есть общая сторона. Однажды, найдя ночлег на поле *A* он задумался, за какое наименьшее количество ходов он сможет добраться до поля *B*. Помогите Его Величеству Белому Королю В Изгнании решить эту псевдошахматную задачку.

#### Формат входного файла

Во входном файле содержатся два натуральных числа *A* и *B*, не превосходящие  $10^6$ .

#### Формат выходного файла

В выходной файл выведите минимальное число ходов, необходимое королю-скитальцу, чтобы достигнуть поля *B*.

#### Пример

<b>a.in</b>	<b>a.out</b>
2 16	6

## Разбор задачи А. Треугольный король

Заметим, что в каждой строке треугольного поля крайняя правая ячейка имеет номер, являющийся полным квадратом.

Благодаря этому, по данному числу  $A$  несложно вычислить его координаты в “треугольной” системе координат. Только введем не две, а три оси, соответствующие трем направлениям, перпендикулярным сторонам треугольников. Все три координаты вычисляются несложно.

Найдем теперь длину кратчайшего пути между клетками  $(x_A, y_A, z_A)$  и  $(x_B, y_B, z_B)$  в этой системе координат. Сколько горизонтальных линий пересечет король на этом пути? Как минимум  $|x_A - x_B|$ , так как при пересечении горизонтальной каждой прямой координата  $x$  меняется на 1. Аналогично, прямые двух других направлений необходимо пересечь минимум  $|y_A - y_B|$  и  $|z_A - z_B|$  раз соответственно.

И оказывается, длина пути в точности равняется  $|x_A - x_B| + |y_A - y_B| + |z_A - z_B|$ . Действительно, если делать только осмысленные ходы, то есть если каждый раз строго уменьшать одно из этих слагаемых (убедитесь, что всегда есть такой ход), то в итоге король окажется в клетке  $B$ .

## Задача В. Палатка

Имя входного файла: **b.in**

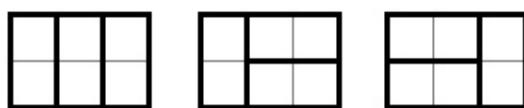
Имя выходного файла: **b.out**

Ограничение по времени: 2 с

Ограничение по памяти: 256 Мб

Будем для упрощения считать, что человек, залезший в спальник и спящий в палатке, занимает на полу прямоугольник  $1 \times t$ . Соответственно, пол классической  $n$ -местной палатки имеет форму прямоугольника  $n \times t$ , и имеется в виду, что люди ложатся параллельно стороне, длина которой равна росту человека.

Однако из практики известно, что есть и другие, более изощренные, способы разместиться  $n$  людям в  $n$ -местной палатке. Ваша задача — сосчитать их количество. Способы, отличающиеся друг от друга симметрией и поворотом, считаются различными, например, при  $t = 2$  есть 3 способа разместить трех человек в трехместной палатке:



## Формат входного файла

Во входном файле содержатся натуральные числа  $m$  и  $n$  ( $2 \leq m \leq 10$ ;  $1 \leq n \leq 40$ ).

## Формат выходного файла

В выходной файл выведите количество способов разместить  $n$  человек в  $n$ -местной палатке.

## Пример

<b>b.in</b>	<b>b.out</b>
2 3	3
4 4	2

## Разбор задачи В. Палатка

Рассмотрим человека, который занимает правый верхний угол палатки (мало того, что ему бедняге холодно, а в случае дождя еще и мокро).

Он лежит либо вертикально (как и предполагается по инструкции, прилагающейся к палатке), либо горизонтально. Если вертикально, то положив его, осталось расположить  $n - 1$  человек в прямоугольнике  $(n - 1) \times m$ . Если же горизонтально, то (в этом несложно убедиться) рядом с ним (илиней!) параллельно лежат еще  $m - 1$  человек. И оставшееся место — прямоугольник  $(n - m) \times m$ .

Итак, рекуррентная формула для ответа на задачу об укладке прямоугольников  $1 \times m$  в прямоугольнике  $n \times m$  такова:

$$a_n = a_{n-1} + a_n - m.$$

## Задача С. Чемпионат по грибному спорту

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

В это воскресенье в Харькове прошел чемпионат студенческих команд по собиранию грибов на заснеженных территориях. Пятеро студентов одного профильного вуза решили накануне потренироваться в парке. Было проведено десять тренировочных забегов, во время каждого из которых двое студентов в течение часа собирали грибы. При этом каждый студент бегал в паре с каждым ровно один раз.

В распоряжении тренера команды теперь имеются десять чисел — количество грибов, собранных каждой парой студентов за час поиска. Он сделал следующее предположение: вероятно, каждый студент за час собирает строго определенное неотрицательное количество грибов, а результат двух студентов — не что иное как сумма их результатов, то есть работа в паре никак не мешает и не помогает.

Определите может ли гипотеза тренера оказаться верной, и если да, то выясните результаты всех пятерых студентов, если бы они бегали по одному.

### **Формат входного файла**

В входном файле содержится десять целых чисел, лежащих в интервале от 0 до  $10^6$ . Это количество грибов, собранных парами студентов во время забегов. Про порядок следования чисел ничего не известно.

### **Формат выходного файла**

В выходной файл выведите слово “WRONG”, если гипотеза тренера точно ошибочна. В противном случае выведите пять строчек с результатами студентов-одиночек в неубывающем порядке.

## Пример

<b>c.in</b>	<b>c.out</b>
11	1
101	10
110	100
1001	1000
1010	10000
1100	
10001	
10010	
10100	
11000	
1	WRONG
2	
3	
4	
5	
6	
7	
8	
9	
10	

## Разбор задачи С. Чемпионат по грибному спорту

Пусть тренер прав, и одиночные результаты студентов, упорядоченные по неубыванию —  $a, b, c, d, e$ .

Во входном файле даны их десять попарных сумм в некотором порядке. К счастью, некоторые из них удается идентифицировать: минимальная попарная сумма это  $a+b$ , максимальная —  $d+e$ , вторая снизу по величине —  $a+c$ , а вторая сверху по величине —  $c+e$  (убедитесь в этом).

Кроме того, сумма всех десяти попарных сумм равна  $(a+b) + \dots + (d+e) = 4a + 4b + 4c + 4d + 4e$ , а значит, сумма  $a+b+c+d+e$  вычисляется как сумма чисел во входном файле, деленная на четыре (кстати, если она на четыре не делится, то тренер точно ошибся).

Поехали:

$$c = (a+b+c+d+e) - (a+b) - (c+d)$$

$$a = (a+c) - c$$

$$b = (a+b) - a$$

$$e = (c + e) - c$$
$$d = (d + e) - e$$

Теперь осталось проверить, что набор попарных сумм этих чисел совпадает (как мультимножество) с набором попарных сумм, данных во входном файле. Если не совпадает, то тренер ошибся.

А если совпадает, то полученные пять чисел и есть ответ. Поверили? Салочка за доверчивость: пять искомых чисел должны быть неотрицательными, и это необходимо отдельно проверить!

### Задача D. Виннимобиль

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

Виннимобиль — новейшее, высокотехнологичное, экологически сверхчистое транспортное средство, работающее на липовом меде. Липовый мед, как известно, чрезвычайно энергетически выгодное топливо: его расход составляет 1 баррель на милю. Бак виннимобиля как раз вмещает 1 баррель меда. Перевозить мед в других контейнерах запрещается: есть опасность, что водитель его просто-напросто съест. Зато разрешается выливать мед из бака на обочину дороги, чтобы потом его собрать и запихать обратно (что самое интересное, без потерь).

Винни хочет добраться на виннимобиле до Пятачка, живущего на расстоянии  $x$  миль, потратив, естественно, минимальное количество меда. Так сколько же баррелей потратит Винни?

#### Формат входного файла

Во входном файле содержится вещественное число  $x$  ( $0 \leq x \leq 5$ ).

#### Формат выходного файла

Выведите количество баррелей меда, необходимое Винни, с точностью до трех знаков после запятой.

#### Пример

<b>d.in</b>	<b>d.out</b>
0.566	0.566
2.000	7.673

## Разбор задачи D. Виннимобиль

Как далеко можно уехать, имея изначально 1 баррель меда?

Только на 1 милю.

Пусть в начале у нас имеется  $y$  — от 1 до 2 — баррелей меда. (Да, будем решать “обратную” задачу: имея  $y$  баррелей, насколько далеко можно уехать?)

Придется поступить так: загрузиться медом, проехать расстояние  $x$ , выгрузить  $1 - 2x$  баррелей меда, и на оставшееся — вернуться на старт. Затем загрузить оставшиеся  $y - 1$  баррелей, проехать  $x$  миль, и вот Винни оказался в точке  $x$ , имея суммарно  $(y - 1 - x) + (1 - 2x) = y - 3x$  баррелей меда.

Если эта величина равна 1, то таким образом можно проехать  $x + 1$  миль. Для этого следует выбрать  $x = \frac{y-1}{3}$ . Итак, проехать можно  $1 + \frac{y-1}{3}$  миль, а при  $y = 2 - 1 + \frac{1}{3}$  мили.

Пусть теперь  $y$  от 2 до 3. Тогда следует сначала довести в некоторую точку  $x$  2 барреля меда, после чего проехать четыре третьих мили описанным выше образом.

Перевести эти 2 барреля можно в три поездки. Первые две позволяют перевезти на расстояние  $x$  два раза по  $1 - 2x$  баррелей, а третья —  $(y - 2 - x)$ , итого в точке  $x$  Винни окажется с  $(y - 2 - x) + 2(1 - 2x) = y - 5x$  баррелями. Оптимально, чтобы это число равнялось 2. Это так, если  $x = \frac{y-2}{5}$ . И тогда максимальный путь Винни составляет  $1 + \frac{1}{3} + \frac{y-2}{5}$ .

Дальнейшие рассуждения аналогичны. Итог их таков: первый баррель меда позволяет передвигаться вперед с расходом  $1 \frac{\text{баррель}}{\text{миля}}$ , следующий баррель —  $\frac{1}{3} \frac{\text{баррель}}{\text{миля}}$ , ...,  $n$ -й баррель — с расходом  $\frac{1}{2n-1} \frac{\text{баррель}}{\text{миля}}$ .

Осталось найти на соответствующем графике точку с ординатой  $x$  (данной во входном файле). См. учебник по программированию, тема “Циклы”.

## Задача E. Схемы рифмовки

Имя входного файла: e.in

Имя выходного файла: e.out

Ограничение по времени: 2 с

Ограничение по памяти: 256 Мб

Рассмотрим строфи стихотворного текста, состоящую из  $n$  строчек. В ней все строчки делятся на несколько классов, внутри каждого из которых строки рифмуются между собой. В данной задаче будем рассматривать только такие строфы, в которых каждая строка рифмуется хотя бы с одной другой.

Например, для катренов (четверостиший) есть четыре возможных схемы рифмовки, удовлетворяющих условиям предыдущего абзаца. Это схемы ААВВ, АВАВ, АВВА и АААА.

Для заданного размера строфы сосчитайте количество различных рифмовых схем.

### **Формат входного файла**

В входном файле содержится целое положительное число  $n$ , не превосходящее 26.

### **Формат выходного файла**

В выходной файл выведите количество схем рифмовки, применимых к  $n$ -стишьям.

### **Пример**

<b>e.in</b>	<b>e.out</b>
4	4

## **Разбор задачи Е. Схемы рифмовки**

Проверим алгеброй гармонию... Пусть  $a_{n,k}$  — количество рифмовых схем для  $n$ -стишия, в которых присутствует ровно  $k$  “классов” рифмующихся строчек.

Приведем рекуррентную формулу:

$$a_{n,k} = (n - 1)a_{n-2,k-1} + ka_{n-1,k}$$

Разоблачим вышеприведенную магию. Рассмотрим  $n$ -стишие, в котором строчки разбиваются ровно на  $k$  классов. Пусть первая строчка попала в класс  $A$ . Рассмотрим два варианта.

Первый — в классе  $A$  ровно две строчки. Тогда есть  $n - 1$  вариантов, какая строчка будет рифмоваться с первой. Зафиксируем один из вариантов. Теперь осталось (музы поэзии, помогите нам) зарифмовать оставшиеся  $n - 2$  строчки так, чтобы они разбились ровно на  $k - 1$  класс. И сделать это ровно  $a_{n-2,k-1}$  способов.

Второй вариант — в классе  $A$  больше двух строчек. Тогда давайте (Каллиопа! Эрато!) зарифмуйем все строчки кроме первой (их  $n - 1$ ), разбив их ровно на  $k$  классов — способов сделать это  $a_{n-1,k}$ . А теперь выберем, к какому из классов “пририфмовать” (Эвтерпа!) первую строчку: сделать это  $k$  вариантов.

## Задача F. Шахтеры

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

На двух угольных шахтах работают шахтеры, производительность которых напрямую зависит от разнообразности еды, которую они потребляют.

Еда приходит на шахты коробками и бывает трех типов: мясо, рыба и хлеб (да-да, вы можете поностальгировать, если десять лет назад играли в “Settlers II”). Когда на шахту приходит очередная коробка с едой, шахтеры добывают некоторое количество угля, зависящее от содержимого этой коробки, а также от двух предыдущих коробок, которые они получали (или меньше чем двух, если они столько еще не получили), следующим образом:

- Если все эти три (или меньше) коробки одного типа, они добывают одну единицу угля;
- Если среди этих трех (или меньше) коробок два типа еды, они добывают две единицы угля;
- Если среди этих трех коробок все три типа еды, они добывают три единицы угля.

Вам заранее известно, в какой последовательности коробки будут поставляться в шахтерский городок. Ваша задача — распределить эту последовательность по двум шахтам так, чтобы суммарное количество добытого угля было максимальным.

Коробки нельзя “откладывать на потом”, а также делить на части: пришедшая коробка должна быть тут же отправлена либо на первую шахту, либо на вторую. Число коробок, доставленных на каждую из шахт может быть произвольным, вплоть до того, что все коробки можно отправить на одну шахту, если это выгодно для дела. (Кстати, если вы в будущем станете госслужащими, так поступать как раз не надо!)

### Формат входного файла

В первой строке входного файла содержится число  $n$  — количество коробок еды ( $1 \leq n \leq 100\,000$ ), во второй — строка длины  $n$ , состоящая из символов “М”, “Р” и “Х”, обозначающих, соответственно, коробку с мясом, рыбой и хлебом, и приведенных в том порядке, в котором коробки будут поставляться.

## Формат выходного файла

Выведите одно число — максимальное количество угля, которые смогут добыть шахтеры в данных условиях.

## Пример

<b>f.in</b>	<b>f.out</b>
6 MBMFFB	12
16 MMVMVBBVMMMBMB	29

## Пояснение

В первом примере можно на каждую шахту доставить по одной коробке каждого типа (например, первую, вторую и четвертую коробки отправив на одну шахту, остальные — на другую), тогда на каждой шахте будет добыто  $1 + 2 + 3 = 6$  единиц угля.

## Разбор задачи F. Шахтеры

Принципиальных состояний у каждой шахты не так много — важны последние полученные две коробки (если такие были). Таких пар (и меньше чем пар) 13 штук:  $((), (M), (F), (B), (MM), (MF), (MB), (FM), (FF), (FB), (BM), (BF), (BB))$ .

Заметим также, что состояния  $(M)$  и  $(MM)$ , а также две другие аналогичные пары — эквивалентны (значение имеет только последняя коробка каждого типа). Остается 10 различных состояний.

Тогда рассмотрим величину  $a_{ijk}$  — максимальное количество угля, которое можно добыть после получения первых  $i$  коробок из последовательности, так чтобы первая шахта пребывала в состоянии  $j$ , а вторая — в состоянии  $k$ .

Из каждого вычислительного состояния есть два перехода: при получении очередной коробки она отправляется либо на одну, либо на другую шахту. Новое состояние этой шахты после этого определяется несложно.

Ответом является максимальное по всем  $j$  и  $k$  значение  $a_{njk}$ .

## Задача G. Подарок Пятачку

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

Однажды Пятачок в приватной беседе рассказал Винни-Пуху, что если есть квадратик  $2 \times 2$ , и в каждую его клетку кладут не более одного желудя, то может получиться либо красиво, либо некрасиво. Более того, Пятачок перечислил все красивые картинки.

Через некоторое время Винни решил сделать Пятачку подарок — прямоугольник высотой  $m$  и шириной  $n$  с желудями, в котором все квадратики  $2 \times 2$  — красивые. Помогите Винни-Пуху сосчитать, сколько существует различных способов сделать это.

### Формат входного файла

В первой строке входного файла содержатся натуральные числа  $m$  и  $n$ , ( $2 \leq m \leq 8$ ;  $2 \leq n \leq 100$ ). Во второй строке — натуральное число, количество красивых квадратиков. Затем идут квадратики, описание каждого — две строки, по два числа в каждой: 1 означает желудь, 0 — его отсутствие. Все квадратики различные.

### Формат выходного файла

Выведите количество прямоугольников, в которых все квадратики — хорошие. Гарантируется, что ответ не превосходит  $10^9$ .

### Пример

<b>g.in</b>	<b>g.out</b>
8 8	
2	
0 1	
1 0	
1 0	
0 1	

## Разбор задачи G. Подарок Пятачку

Это задача решается методом динамического программирования по профилю. В том числе, по изломанному профилю.

Профилем в данном случае будем называть информацию о том, как был заполнен очередной столбец. (Изломанным профилем — информацию о том, как был заполнен “изломанный столбец”).

Два профиля являются совместимыми, если все квадратики  $2 \times 2$ , получающиеся при составлении профилей рядом, являются красивыми.

В первом слое следует расставить единицы во всех ячейках: для любого профиля  $r$  существует ровно один способ заполнить первый столбец, получив профиль  $r$ .

Ответом является сумма по всем ячейкам последнего столбца: красивое заполнение всего прямоугольника, с каким угодно профилем в последнем слое, безусловно, устраивает Пятачка.

### **Задача Н. Игральная кость**

Имя входного файла:	<code>h.in</code>
Имя выходного файла:	<code>h.out</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

На клетчатом поле размером  $m \times n$  в левом нижнем углу лежит игральная кость. За один ход ее можно перекатить на клетку вправо или вверх. Стоимостью пути называется сумма чисел на верхней грани кубика во всех клетках пути (включая начальную и конечную).

Найдите минимальную стоимость пути в правый верхний угол.

#### **Формат входного файла**

В первой строке два натуральных числа  $m$  и  $n$  ( $1 \leq m, n \leq 1000$ ) — ширина и высота доски. Во второй строке три числа от 1 до 6 — числа на верхней, левой и передней грани кубика соответственно. Сумма чисел на противоположных гранях кубика равна 7, все числа на гранях кубика различны.

#### **Формат выходного файла**

Выведите минимальную возможную стоимость пути.

#### **Пример**

<code>h.in</code>	<code>h.out</code>
6 1 1 2 3	17
5 5 1 4 5	29

## Разбор задачи Н. Игровая кость

Автор благодарит Виктора Викторовича Кроппа за работу над данной задачей.

Задача решается динамическим программированием, где состоянием является пара: координаты игральной кости на доске + расположение игральной кости.

Из каждого состояния существует два перехода (в общем случае), соответствующих перекатыванию кости вверх и вправо. Новые координаты получаются тривиально (“+1” к одной из них), а новое состояние — несложное и полезное упражнение для пространственного воображения.

Возможных расположений кости — 24. (Верхней гранью может быть любая из шести граней, а когда верхняя грань зафиксирована, на роль передней грани есть четыре кандидата). Благодаря этому, можно, например, вручную задать массив новых расположений при переходе из каждого расположения вверх и вправо.

Отметим изящный (в плане простоты реализации) подход, допустимый благодаря тому, что в данной задаче гарантируется, что сумма цифр на противоположных гранях равна 7, а сами цифры от 1 до 6. Будем кодировать расположение кости тремя числами — теми, что написаны при данном расположении на верхней, левой и передней гранях кубика. (В частности, начальное положение задано во входном файле именно в таком виде.) Тогда перекатывание кости вправо есть переход  $(a, b, c) \rightarrow (b, 7 - a, c)$ , а перекатывание вверх есть переход  $(a, b, c) \rightarrow (c, b, 7 - a)$ .

Добавим, что если бы были разрешены ходы не только вправо и вверх, то данная задача бы состояла в поиске кратчайшего пути в графе, вершинами которого является то, что описано выше как состояния в динамическом программировании. Эта задача решается, например, алгоритмом Дейкстры.

## Задача I. Пятизвездочная задача

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

В изначально пустой прямоугольник разрешается ставить пять звездочек в ряд по вертикали или горизонтали, не выходя при этом за границы прямоугольника. При этом различные пятерки звездочек имеют право пересекаться и налегать друг на друга.

Вам дано расположение звездочек в прямоугольнике. Можно ли его получить по указанным правилам, и если можно, то какое минимальное количество пятерок звездочек надо поставить, чтобы его получить?

### Формат входного файла

В первой строке входного файла содержатся два числа  $m$  и  $n$  — высота и ширина прямоугольника ( $1 \leq m \leq 5$ ;  $1 \leq n \leq 10$ ).

В каждой из следующий  $m$  строк содержится по  $n$  символов “\*” и “.”, соответствующих звездочке и пустой клетке.

### Формат выходного файла

Выведите -1, если данную конфигурацию получить невозможно. В противном случае выведите наименьшее число пятерок звездочек, с помощью которого можно получить данную конфигурацию.

### Пример

i.in	i.out
5 6 .*.... .***** . *.... *****. . *....	3
3 10 ...*****.. .*****.*. .*****.*.*	5
5 6 ..... .****. .****. .****. .****.	-1

### Разбор задачи I. Пятизвездочная задача

Несмотря на то, что решение с использованием метода динамического программирования существует, опишем более элегантное решение, правда, допустимое лишь при имеющихся (небольших) ограничениях.

Так как высота данного прямоугольника не превышает 5, существует всего не более 10 возможных позиций для вертикальных пятерок звездочек — по одной в каждом столбце, и то, только если высота в точности равна 5.

Переберем все варианты (их не более  $2^{10}$ ), какие из возможных вертикальных пятерок были поставлены. Рассмотрим только те, что не противоречат конфигурации, данной во входном файле.

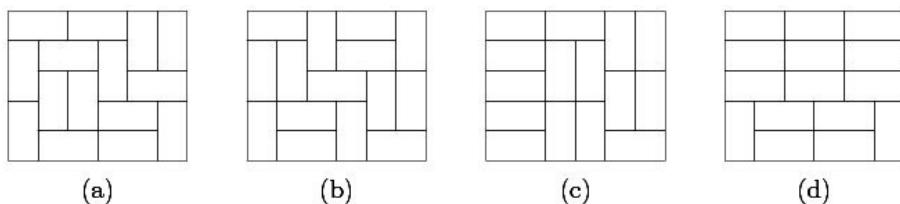
Теперь, когда вертикальные пятерки поставлены, осталось провести горизонтальные, чтобы покрыть все еще не покрытые звездочки. Эту задачу можно решить, во-первых, отдельно в каждой строке, а во-вторых, жадным алгоритмом: найдем самую левую непокрытую звездочку и проведем через нее горизонтальную пятерку, сдвинув ее вправо на столько, на сколько это возможно.

### Задача J. Прочные замощения

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

Замощение прямоугольника  $m \times n$  доминошками  $2 \times 1$  будем называть прочным, если не существует прямой, пересекающей внутренность прямоугольника  $m \times n$  и не пересекающей внутренность ни одной доминошки.

Например, приведенные на иллюстрации замощения (a) и (b) — прочные, а замощения (c) и (d) — нет.



А сколько существует прочных замощений прямоугольника  $m \times n$ ?

#### Формат входного файла

В первой строке два натуральных числа  $m$  и  $n$  ( $1 \leq m \leq 8$ ;  $1 \leq n \leq 16$ ) — ширина и высота доски.

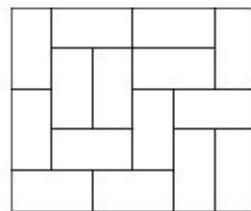
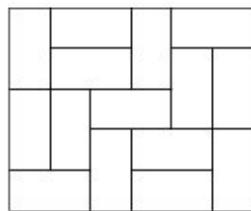
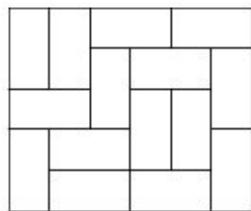
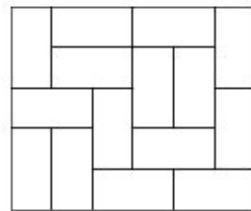
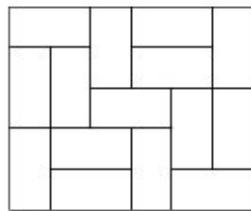
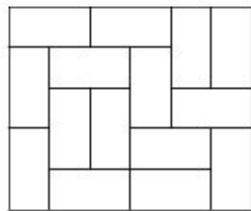
#### Формат выходного файла

Выведите одно число — количество прочных замощений данного прямоугольника.

## Пример

j.in	j.out
2 2	0
5 6	6

**Пояснение** Приведем все прочные замощения прямоугольника  $5 \times 6$ :



## Разбор задачи J. Прочные замощения

Автор благодарит Андрея Сергеевича Станкевича за предоставленную задачу.

Задача решается динамическим программированием по изломанному профилю. Профиль включает следующую информацию: во-первых, в каких позициях доминошки пересекают границу (это  $n + 1$  бит, а если выбрать правильный подход, то  $n$  бит), во-вторых, какие горизонтальные линии между клетками уже были хотя бы раз “перекрыты” вертикальными доминошками (это  $n - 1$  бит).

Из каждого состояния динамического программирования не более двух переходов. Всего состояний  $O(n \cdot n \cdot 2^{2n-1})$ .

Отсутствие горизонтальных “непрочностей” обеспечивается тем, что в конце будут прибавлены к ответу только те замощения, в которых все  $n - 1$  горизонтальных линий были “перекрыты” хотя бы раз.

А вертикальные “непрочности” ликвидируются следующим образом: если изломанный профиль имеет вид “неизломанного”, то есть заполнено некоторое количество столбцов целиком, то профиль, в котором нет доминошек, пересекающих границу (то есть с маской пересечений 0), должен

считаться некорректным, ведь он означает идеально ровно заполненные несколько столбцов, а это и есть вертикальная “непрочность”. Исключение составляет, конечно, момент, когда весь прямоугольник  $m \times n$  оказался замощен — тогда наоборот, только профиль без “торчащих” доминошек и рассматривается.

### **Задача К. Гамильтонов трубопровод**

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

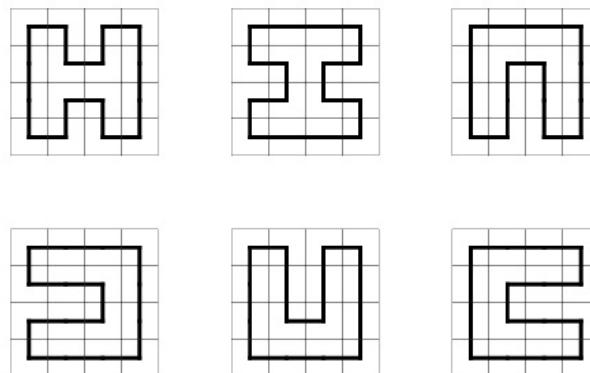
Город Намхеттен является прямоугольником  $m \times n$ , состоящим из квадратных кварталов.

Инженер Гамильтон хочет построить в этом городе водопровод, представляющий собой замкнутую трубу, проходящую под каждым из кварталов ровно по одному разу.

Под каждым кварталом труба может либо проходить по прямой, либо изгибаться на 90 градусов.

Сколько существует планов Гамильтонова водопровода?

Например, для города размером  $4 \times 4$  существует 6 планов:



### **Формат входного файла**

Во входном файле содержатся натуральные числа  $m$  и  $n$  — размеры Намхеттена ( $1 \leq m \times n \leq 100$ ).

### **Формат выходного файла**

Выведите одно число — количество планов Гамильтонова водопровода, осуществимых в данном городе.

## Пример

<b>k.in</b>	<b>k.out</b>
4 4	6
5 7	0
2 8	1
12 8	102283239429

## Разбор задачи К. Гамильтонов трубопровод

Автор благодарит Андрея Сергеевича Станкевича за предоставленную задачу.

Решается она методом динамического программирования по изломанному профилю, и профиль этот весьма нетривиален.

Он содержит, во-первых, конечно, информацию о том, в какой точке границы имеется “торчащая” труба, а в какой — нет. Если высота прямоугольника —  $n$ , то длина границы профиля —  $n + 1$ , и именно столько бит требуется, чтобы закодировать информацию о трубах.

Легко понять, что “торчащих” концов труб четное число (ведь у каждого участка трубы два конца). Эти концы труб разбиваются на пары, соответствующие одному связному участку трубы. Так вот, информацию об этих парах необходимо хранить в профиле.

Важным является наблюдение о том, что эти пары концов труб образуют правильную скобочную последовательность. Действительно, если рассмотреть трубу, идущую от границы профиля как-то через весь прямоугольник, и возвращающуюся к границе, то любая другая труба, имеющая два выхода к границе профиля (равно как и любая вообще) будет находиться либо “внутри” данной трубы, либо “снаружи”.

А это означает, что можно кодировать “открывающие скобки” как единицы, а “закрывающие” — как двойки. А имея последовательность (корректную!) из единиц и двоек, можно однозначно определить, какой конец трубы соответствует какому.

Итого, профиль есть вектор из  $n + 1$  числа, каждое из которых — от 0 до 2. Причем, далеко не каждый такой профиль является допустимым.

Правила перехода в динамике по изломанному профилю не столь сложны. Единственный особый случай — когда замыкаются воедино два “торчащих” конца труб. В этом случае есть два важных варианта: если они не соответствуют друг другу, то они “замыкаются”, а концы, соответствовавшие раньше этим двум, теперь начинают соответствовать друг другу.

Если же они соответствуют как раз друг другу, то их замыкание допустимо ровно в одном случае: если это правый нижний угол прямоугольника. Тогда это замыкание дает Гамильтонов цикл. Во всех остальных случаях замыкание дало бы цикл, проходящий не по всем клеткам прямоугольника, а это противоречит наличию Гамильтонова цикла.

### **Задача L. Шестиугольник и ромбические домино**

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

Шестиугольник с стороной  $n$  разбит на  $6n^2$  правильных треугольников со стороной 1.

Сколькими способами его можно покрыть ромбическими домино (без наложений и выходов за границу)?

(Ромбическое домино состоит из двух правильных треугольников со стороной 1, смежных по стороне.)

#### **Формат входного файла**

Во входном файле содержится число  $n$  ( $1 \leq n \leq 7$ ).

#### **Формат выходного файла**

Выведите количество замощений шестиугольника.

#### **Пример**

l.in	l.out
2	20
1	2

### **Разбор задачи L. Шестиугольник и ромбические домино**

Автор благодарит Андрея Сергеевича Станкевича за предоставленную задачу.

Решением является динамическое программирование по изломанному профилю.

Состояние  $(i, j, k)$  — ситуация, когда первые  $i$  “столбцов” заполнены целиком, в  $(i+1)$ -м столбце заполнено  $j$  верхних треугольников, и профиль “торчащих” ромбических домино кодируется битовой маской  $k$ .

Из каждого состояния существует не более трех переходов, соответствующих расположению в заполняемом треугольнике ромбического домино одного из трех возможных направлений.

Всего состояний  $O(n \cdot n \cdot 2^{2n})$ , переходов — столько же.

## Задача М. Симпатичные узоры 2

Имя входного файла: `m.in`

Имя выходного файла: `m.out`

Ограничение по времени: 5 с

Ограничение по памяти: 256

Компания *BrokenTiles* планирует заняться выкладыванием во дворах у состоятельных клиентов узор из черных и белых плиток, каждая из которых имеет размер  $1 \times 1$  метр. Известно, что дворы всех состоятельных людей имеют наиболее модную на сегодня форму прямоугольника  $n \times m$  метров.

Однако при составлении финансового плана у директора этой организации появилось целых две серьезных проблемы: во первых, каждый новый клиент очевидно захочет, чтобы узор, выложенный у него во дворе, отличался от узоров всех остальных клиентов этой фирмы, а во вторых, этот узор должен быть симпатичным.

Как показало исследование, узор является симпатичным, если в нем нигде не встречается квадрата  $2 \times 2$  метра, полностью покрытого плитками одного цвета.

Для составления финансового плана директору необходимо узнать, сколько клиентов он сможет обслужить, прежде чем симпатичные узоры данного размера закончатся. Помогите ему!

### Формат входного файла

На первой строке входного файла находятся два натуральных числа  $n$  и  $m$ .  $1 \leq n \cdot m \leq 300$ .

### Формат выходного файла

Выведите в выходной файл единственное число — количество различных симпатичных узоров, которые можно выложить во дворе размера  $n \times m$  по модулю  $2^{30} + 1$ . Узоры, получающиеся друг из друга сдвигом, поворотом или отражением считаются различными.

## Пример

<b>m.in</b>	<b>m.out</b>
2 2	14
3 3	322

## Разбор задачи М. Симпатичные узоры 2

Автор благодарит Сергея Копелиовича за предоставленную задачу.

Задача решается методом динамического программирования, причем при данных ограничениях — исключительно по изломанному профилю.

Профиль в данном случае хранит  $n + 1$  бит — информацию о цветах соответствующих клеток.

При переходе от состояния к состоянию динамического программирования следует решить вопрос о раскраске очередной ячейки в один из двух цветов — это можно сделать тогда и только тогда, когда квадрат  $2 \times 2$ , в котором эта ячейка является правой нижней клеткой, не является одноточечным.

Таким образом, переход от состояния к состоянию прост и осуществим за время  $O(1)$ . Суммарное же время работы алгоритма составляет  $O(mn2^n)$ .

## **День четвертый (15.02.2011 г.)**

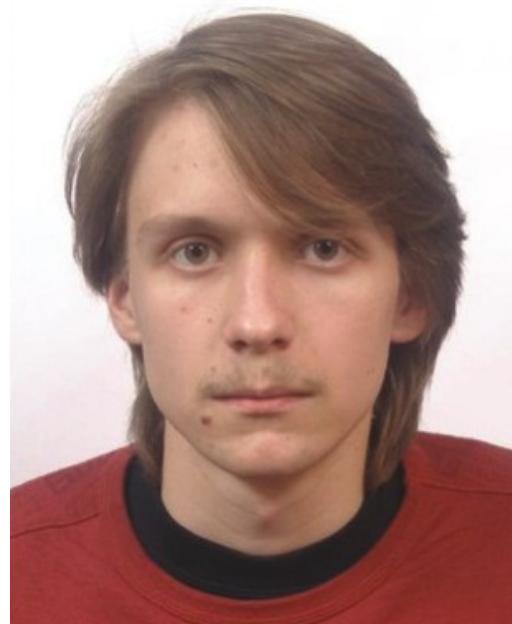
### **Конкурс Киевского Политехнического Института**

#### **Об авторах...**

**Кордубан Дмитрий Александрович**,  
родился в 1988 году. Закончил Национальный технический университет Украины “КПИ” в 2010 году. Интересуется наукой, в частности: распознаванием образов, машинным обучением.

Основные достижения:

- участник финала ACM ICPC 2009 в составе команды КПИ;
- победитель Всеукраинской студенческой олимпиады 2009 года в составе команды КПИ;
- в сентябре 2010 года на финальных соревнованиях Всеукраинской студенческой олимпиады по программированию команда КПИ, которую он тренировал, заняла призовое третье место;
- с 2006 года заместитель главы жюри Киевской городской олимпиады школьников по информатике.



**Слюсаренко Алексей Александрович**,  
родился 1990 году. Учится на 4 курсе  
Национального технического университета  
Украины “КПИ” на специальности “систем-  
ный анализ и управление”.

Основные достижения:

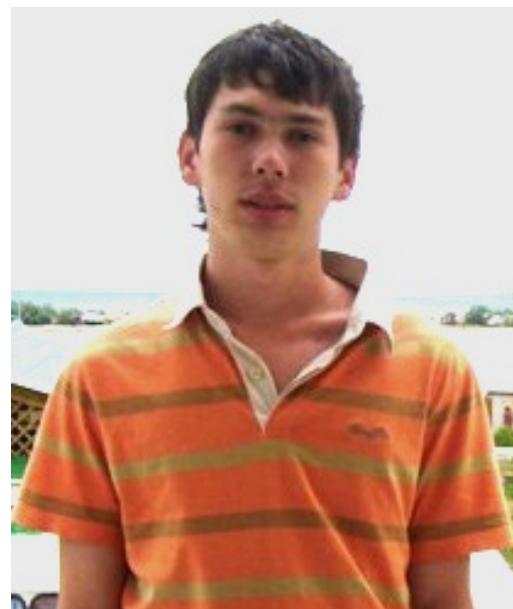
- участник финала ACM ICPC 2009 в составе команды КПИ;
- победитель Всеукраинской студенческой олимпиады 2009 года, 3-е место 2010 года – в составе команды КПИ;
- 11-е место в финале Google Code Jam 2009.



**Костеев Евгений Евгеньевич**, родился в 1988 году. Учится на 6 курсе Национального технического университета Украины “КПИ” специальности “системный анализ финансового рынка”.

Основные достижения:

- 5-е место на SEERC 2009 в составе команды КПИ;
- 3-е место Всеукраинской студенческой олимпиады 2010 в составе команды КПИ.



## Теоретический материал. Вероятностные структуры данных

### **Мотивация**

Хеш-таблицы — это очень практические вероятностные структуры данных, позволяющие реализовать за время  $O(1)$  в среднем словарные операции:

- Поиск  $\text{Lookup}(x)$ : возвращает `true`, если  $x$  содержится в множестве, и `false` в противном случае.
- Вставка  $\text{Insert}(x)$ : добавляет элемент  $x$  в множество, если его там не было.
- Удаление  $\text{Delete}(x)$ : удаляет  $x$  из множества, если он там присутствовал.

При практическом использовании хеш-таблиц могут возникать различные дополнительные требования, такие как константное время работы определенного запроса в худшем случае или ограниченная память. Мы рассмотрим вероятностные структуры данных, ориентированные под такие задачи.

### **Допущения**

Все приведенные алгоритмы используют одну или несколько *хеш-функций*, сопоставляющие элементам „случайные” числа из множества  $\{0, 1, \dots, m - 1\}$ . Для простоты предположим, что:

1. Размер элементов ограничен константой. Мы можем сравнивать два элемента за константное время.
2. У нас есть доступ к произвольному количеству хеш-функций  $h_1, h_2, \dots, h_k, \dots$  таких, что  $h_i(x)$  принимает какое-то конкретное значение из  $\{0, 1, \dots, m - 1\}$  с вероятностью  $1/m$ , и эти функции независимы. Хеш-функции могут быть вычислены за константное время.
3. Количество элементов, хранимых в множестве, ограничено сверху числом  $n$ .

Второе допущение сделано для простоты анализа и на практике может быть ослаблено. Заметим, что оно может быть выполнено только тогда, когда сами хеш-функции выбираются случайно из некоторого семейства. Этот вопрос будет подробнее освещен в последнем подразделе.

Мы будем требовать, чтобы используемая структурами данных память была порядка  $O(n)$ , т.е. асимптотически оптимальной.

### **Хеширование с цепочками**

Основная идея хеш-таблиц состоит в том, что значение хеш-функции определяет место хранения каждого элемента. Например, можно пытаться хранить элемент  $x$  в позиции массива  $h(x)$ , если  $m \geq n$ . Однако такой подход с большой вероятностью приведет к *коллизиям*, т.е. различным элементам  $x$  и  $y$  для которых  $h(x) = h(y)$ . Очевидное решение состоит в том,

чтобы хранить в позиции указатель на структуру данных, содержащую все элементы с соответствующим значением хеш-функции — например, связный список. Будем также называть эту структуру данных *цепочкой*. Будем говорить также о цепочке  $x$ , подразумевая цепочку в позиции  $h(x)$ .

Отметим два следующих наблюдения:

1. Для любых двух элементов  $x$  и  $y$  вероятность того, что  $x$  попадет в цепочку  $y$  есть  $O(1/m)$ . Это следует из нашего допущения о  $h$ .
2. Время, затрачиваемое на обработку элемента  $x$ , пропорционально количеству элементов в цепочке  $x$ .

Оценим время операции над элементом  $x$ . Согласно наблюдению 2, мы можем сделать это путем оценки ожидаемого размера цепочки  $x$ . Без потери общности можно считать, что элемент  $x$  не содержится в множестве во время наших операций — это только увеличивает их время. Так что пускай в цепочке  $x$  все элементы отличны от  $x$ . Пусть  $S$  — множество хранимых элементов в момент начала операции. Для любого  $y \in S$  наблюдение 1 утверждает, что вероятность того что операция затратит время на обработку элемента  $y$  есть  $O(1/m)$ . Таким образом, ожидаемое время затрачиваемое на конкретный элемент  $y$  есть  $O(1/m)$ . Чтобы получить общее ожидаемое время, просуммируем это по всем элементам  $S$ . Это равно  $|S| \cdot O(1/m) = O(1)$  в силу выбора  $m \geq n \geq |S|$ . Итак, ожидаемое время работы любой операции константно.

### ***Идеальное хеширование***

Иногда множество, которое должно храниться в словаре, фиксировано заранее — требуется лишь обеспечить быстрое выполнение операций поиска  $\text{Lookup}(x)$ . В таком случае можно построить за ожидаемое линейное время структуру данных, отвечающую на такие запросы за  $O(1)$  в *худшем случае*.

Если бы требование о линейных затратах памяти отсутствовало, такую структуру было бы очень легко построить. Для этого достаточно положить  $m = n^2$  в схеме хеширования с цепочками. Тогда вероятность отсутствия коллизий будет больше  $1/2$ . В самом деле, существует  $\binom{n}{2}$  различных пар элементов  $(x, y)$ . Коллизия каждой отдельной пары происходит с вероятностью  $1/m$ , так что общая вероятность наличия коллизий не превосходит  $\binom{n}{2}/m < 1/2$ .

Приведенное рассуждение является в каком-то смысле обращением „парадокса дней рождений“. Таким образом можно выбирать случайные хеш-функции, удовлетворяющие нашим свойствам, и пытаться строить хеш-таблицу, а при появлении коллизий просто брать другую функцию. В сред-

нем нам понадобиться сделать это дважды. Однако как построить идеальный хеш в линейной памяти?

Для этого используем двухуровневую схему. Сначала добавим все элементы в хеш-таблицу (с цепочками) размера  $m = n$ . При этом почти наверняка возникнут коллизии. Однако затем, вместо того чтобы хранить цепочки связными списками, применим к ним описанное только что идеальное хеширование в квадратичной памяти.

Пусть хеш-функция первого уровня отобразила в позицию  $i$  ровно  $n_i$  элементов. Тогда, как было показано ранее, такая двухуровневая конструкция может быть построена с затратами памяти порядка  $\sum_i n_i^2$ . Мы покажем, что с большой вероятностью справедливо  $\sum_i n_i^2 = O(n)$ . Точнее, докажем следующее

$$\Pr\left[\sum_i n_i^2 > 4n\right] < 1/2.$$

Для этого покажем, что  $\mathbf{E}[\sum_i n_i^2] < 2n$ . Тогда требуемое будет следовать из неравенства Маркова. (Если бы вероятность того, что сумма может быть больше  $4n$ , была равна хотя бы  $1/2$ , то из этого следовало бы что матожидание суммы равнялось по крайней мере  $2n$ . Поэтому если матожидание меньше  $2n$ , то вероятность неудачи меньше  $1/2$ .)

Применим следующий трюк: будем интерпретировать сумму квадратов как количество коллизий упорядоченных пар элементов, включая совпадения элементов самих с собой. Так, если цепочка содержит элементы  $\{a, b, c\}$ , то элемент  $a$  совпадет с каждым из элементов  $\{a, b, c\}$ ,  $b$  и  $c$  аналогично — всего 9 совпадений. Итак,

$$\begin{aligned} \mathbf{E}\left[\sum_i n_i^2\right] &= \mathbf{E}\left[\sum_x \sum_y [x \text{ и } y \text{ совпадают}]\right] \\ &= n + \sum_x \sum_{y \neq x} \mathbf{E}[x \text{ и } y \text{ совпадают}] = \\ &= n + n(n-1)/m \quad (1/m \text{ из свойств хеш-функции}) \\ &< 2n. \quad (\text{так как } m = n) \quad \blacksquare \end{aligned}$$

Таким образом, мы пробуем новую хеш-функцию первого уровня до тех пор, пока не получим  $\sum_i n_i^2 < 4n$ , а затем используем квадратичный метод для хранения каждой цепочки.

Приведенная структура не позволяет вставлять элементы в множество динамически. Но последнее тоже возможно, например с использованием

хеширования кукушки<sup>1</sup>.

### **Фильтрация Блума**

Иногда бывает нужно хранить множество очень больших по размеру объектов. В таком случае линейная оценка памяти хеш-таблиц становится очень непрактичной, т.к. все объекты все равно хранятся в памяти и выполняются долгие операции их сравнения. Однако эту задачу можно решить с использованием лишь константного количества бит памяти на каждый вставленный элемент.

Ценой за такое улучшение служит недетерминированность получаемого ответа: в случае, если какой-то элемент не содержится в множестве, структура данных может с небольшой вероятностью сообщить, что он там содержится. Однако, если элемент на самом деле содержится во множестве, то ответ будет всегда правильным.

Покажем сначала, как реализовать такую структуру без операций удаления. Создадим битовый массив из  $m > n$  элементов и выберем  $k$  различных хеш-функций  $h_1, h_2, \dots, h_k$ . Будем делать следующее: при добавлении элемента  $x$  во множество, установим биты  $h_1(x), h_2(x), \dots, h_k(x)$  в 1. При ответе же на запрос  $\text{Lookup}(x)$  проверим эти биты. Если один из них равен 0, то очевидно что элемент  $x$  отсутствует во множестве. Иначе сообщим, что элемент присутствует.

Какова вероятность ошибки в случае, когда элемент на самом деле отсутствует? Заметим, что после  $n$  вставок вероятность того, что какой-то конкретный бит остался нулевым в точности равна:

$$\left(1 - \frac{1}{m}\right)^{kn}.$$

Для возникновения ошибки необходимо, чтобы каждый из  $k$  битов соответствующих запросу был выставлен в 1, вероятность этого равна:

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.$$

Правая часть достигает минимума при  $k = \ln 2 \cdot m/n$ , обеспечивая вероятность ошибки

$$\left(\frac{1}{2}\right)^k \approx (0.6185)^{m/n}.$$

Таким образом, нам достаточно использовать примерно 14.4 бит на каждый вставленный элемент, чтобы обеспечить вероятность ошибки не более

---

<sup>1</sup>Rasmus Pagh, Flemming Friche Rodler. Cuckoo Hashing. — Proceedings of the 9th Annual European Symposium on Algorithms, 2001.

$10^{-3}$ , 28.8 бит для вероятности ошибки не более  $10^{-6}$  и так далее — *независимо от размера самих вставляемых объектов*.

Недостатком структуры является невозможность удалений. Это можно исправить, если хранить не один бит в каждой позиции массива, а несколько. Будем трактовать их как счетчики величины „количество вставок минус количество удалений” элементов с определенным значением хеш-функции. Так, 4-битовый счетчик может хранить значения от 0 до 15. При вставке элемента будем увеличивать счетчики в соответствующих позициях, а при удалении уменьшать. Реализация операции  $\text{Lookup}(x)$  остается неизменной.

Поскольку размер счетчиков ограничен, необходимо аккуратно реализовать обработку переполнений. Так, если в приведенном примере счетчик достиг значения 15, то увеличивать его дальше нельзя. Очевидно, что это не нарушит свойств структуры. Тонким моментом является тот факт, что уменьшать его тоже нельзя! Иначе возможна такая последовательность вставок и удалений, при которой счетчик станет равен 0, но в структуре еще останутся элементы, распределенные в эту позицию. Таким образом, как только счетчик достигает максимального значения — он остается в нем навсегда. В результате вероятность ошибки немного увеличивается с ростом количества операций. Точный анализ счетчиков Блума можно найти в *Handbook of Algorithms and Data Structures*<sup>2</sup>.

### **Универсальные семейства хеш-функций**

В самом начале мы предположили, что имеем доступ к „случайным” хеш-функциям, сопоставляющие объектам  $m$  значений с равной вероятностью. Если мы возьмем действительно случайную функцию, то для ее хранения понадобиться по крайней мере  $\Omega(n \log m)$  бит, да и за  $O(1)$  ее нельзя будет вычислить.

Заметим однако, что во всех приложениях нам не требовалась случайность в этом смысле. Единственное, что нужно для хорошей работы хеширования — оценка на вероятность коллизии

$$\Pr [h(x) = h(y)] \leq 1/m$$

для любой пары  $x \neq y$  (вероятность берется по выбору хеш-функции). Семейства хеш-функций, обладающих этим свойством, называются *2-универсальными*.

Оказывается, что есть очень маленькие и удобные 2-универсальные семейства. Например, если объекты — целые числа в диапазоне от 0 до  $n-1$ , то можно поступить следующим образом. Выберем простое число  $p$  между  $n$  и  $2n$ . Тогда искомое семейство функций выглядит так:

---

<sup>2</sup>G. Gonnet, R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. — Addison-Wesley, 1991.

$$H = \{x \mapsto ((ax + b) \mod p) \mod m \mid a, b \in \mathbb{Z}_p, a \neq 0\}.$$

Это семейство хорошо во всех отношениях: описание функции занимает  $O(\log n)$  бит, легко выбирать случайную  $h \in H$ , и функции этого семейства легко вычисляются за  $O(1)$ .

## Задачи и разборы

### Задача А. Инопланетяне

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

Недавно ученые послали в космос летательный аппарат с запиской на борту. В этой записке было указано одно единственное число, которое было сгенерировано случайно. Ровно через месяц на поверхность Земли, в районе Харькова, приземлился ответный летательный аппарат. На его борту была обнаружена аналогичная записка, в которой было указано тоже одно число. Что это число могло бы значить, никто не знает, но есть предположение, что инопланетяне решили перевести полученное ими число в их систему счисления (естественно они догадались, что наше число было записано в десятичной системе счисления) и прислать его нам обратно. Ваша задача состоит в том, чтобы определить, в какой системе счисления прислали ответ инопланетяне.

#### Формат входного файла

Ввод состоит из двух строк. Первая строка входного файла содержит число, отправленное нами в космос. Вторая строка входного файла содержит ответное число. Оба числа состоят из цифр 0 – 9, положительны и без ведущих нулей. Длины чисел не превосходят  $10^5$ .

#### Формат выходного файла

Выведите число, означающее систему счисления, в которой было записано ответное число. Если ответов несколько, выведите минимальный. Если ответа не существует либо не лежит в диапазоне от 2 до 9, выведите 0.

## Пример

a.in	a.out
10 1010	2
9 100	3

## Разбор задачи А. Инопланетяне

Основная проблема данной задачи заключалось в том, чтобы проверять равенство двух чисел, записанных в разных системах счисления. Научившись этому, осталось бы просто перебрать основание второго числа от 2 до 9 и проверить равенство для этого основания.

Предлагается следующий алгоритм решения данной проблемы. Вычислим оба числа  $a$  и  $b$  по модулю случайногоприме  $p$  ( $p$  — простое,  $1 \leq p \leq 10^9$ ),  $x = a \bmod p$ ,  $y = b \bmod p$ . Тогда, в случае  $x \neq y$  можно точно сказать, что числа  $a$  и  $b$  не равны. Если вышло так, что  $x = y$ , то выберем еще одно или два простых числа и выполним аналогичную проверку для них. В случае равенства и по этим модулям, можно сказать, что эти числа равны с некоторой вероятностью. Оценим, какова же эта вероятность.

Рассмотрим  $A = |x - y|$ . Не трудно убедиться, что  $A$  будет делиться на случайное простое число с вероятностью  $\frac{\log_2(A)}{P}$ , где  $P$  — количество простых чисел от 1 до  $10^9$ , так как любое число имеет не больше логарифма простых различных делителей. Учитывая условия данной задачи заданная вероятность приблизительно равна 0.01, что доказывает корректность предложенного алгоритма.

## Задача В. Конфеты

Имя входного файла: b.in

Имя выходного файла: b.out

Ограничение по времени: 3 с

Ограничение по памяти: 256 Мб

Кондитерская фабрика “Квадрошен” выпускала все конфеты в квадратных коробках. К Новому Году руководство компании решило провести акцию — в каждой коробке увеличить количество конфет на 1, при этом сохранив цену. Разумеется, размеры всех коробок необходимо увеличивать. Причём дизайнеры решили, что коробка будет выглядеть ещё больше, если

сделать её периметр максимально возможным при том, что вдоль каждой стороны вмещалось хотя бы 2 конфеты. Понятно так же, что пустых мест в коробке оставаться не должно. Так как фабрика выпускает довольно много различных коробок, Вас попросили написать программу для расчёта их размеров.

### **Формат входного файла**

Первая строка входного файла содержит единственное целое число  $k$  — количество коробок, выпускаемых фабрикой ( $1 \leq k \leq 100\,000$ ). Каждая из следующих  $k$  строк содержит по одному числу  $n_i$  — изначальный размер коробки, то есть количество конфет вдоль стороны ( $1 \leq n_i \leq 1\,000\,000$ ).

### **Формат выходного файла**

Выходной файл должен содержать  $k$  строк. В  $i$ -той строке напечатайте одно число — наибольший возможный периметр  $i$ -той коробки или число 0 в случае, если удовлетворить требованиям дизайнеров не получается.

### **Пример**

<b>b.in</b>	<b>b.out</b>
3	14
3	54
7	0
2	

### **Разбор задачи В. Конфеты**

Из условия данной задачи понятно, что главное задание — найти минимальный делитель для всех чисел вида  $n^2 + 1$ . Для этого можно использовать алгоритм чем-то напоминающий алгоритм решета Эратосфена. Будем хранить ряд чисел вида  $n^2 + 1$  ( $2, 5, 10, 17, 26, 37, 50, 65\dots$ ). Далее будем по очереди выбирать каждое из этих чисел и производить операцию «сокращения». То есть: в начале мы возьмём число  $1^2 + 1 = 2$ , понятно что на 2 будут делиться так же числа  $(1+2)^2 + 1 = 10$ ,  $(1+4)^2 + 1 = 26$ ,  $(1+6)^2 + 1 = 50$  и.т.д. Сократим их на максимальное количество двоек, в итоге получим новый ряд:  $1, 5, 5, 17, 13, 37, 25, 65\dots$  Рассмотрим второе число из ряда — 5. Таким же образом понимаем, что надо сократить числа под номерами 2, 7, 12, 17.... Получим ряд:  $1, 1, 5, 17, 13, 37, 1, 65\dots$  На следующей итерации мы сократим числа в позициях 3, 8, 13, 18... на 5. Получим ряд:  $1, 1, 1, 17, 13, 37, 1, 13\dots$  Можно доказать, что выбранный очередной делитель, на который мы будем сокращать будет либо простым,

либо единицей. То есть, в этом алгоритме мы находим все простые делители чисел вида  $n^2 + 1$ .

Можно оценить сверху время работы данного алгоритма:  
 $n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \dots + \frac{n}{n} \leq n(\ln n + 1)$ . То есть время работы  $O(n \log n)$ .

### Задача С. Раскраска

Имя входного файла: **c.in**

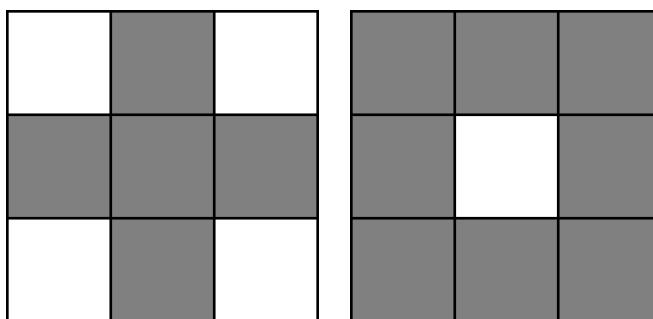
Имя выходного файла: **c.out**

Ограничение по времени: 2 с

Ограничение по памяти: 256 Мб

Прямоугольник размером  $n \times m$  разделен на  $n \cdot m$  единичных квадратов. Ваша задача — раскрасить квадраты в 5 различных цветов так, чтобы:

1. каждая фигурка “крест” была раскрашена во все 5 цветов;
2. каждая фигурка “окно” была раскрашена не более чем в 4 различных цвета.



### Формат входного файла

Первая строка входного файла содержит два целых числа  $n$  и  $m$  — размеры прямоугольника ( $1 \leq n, m \leq 30$ ).

### Формат выходного файла

Выведите  $n$  строк по  $m$  символов в каждой, задающие искомую раскраску. Различные цвета обозначаются цифрами от 1 до 5.

### Пример

<b>c.in</b>	<b>c.out</b>
3 3	123 143 151

## Разбор задачи С. Раскраска

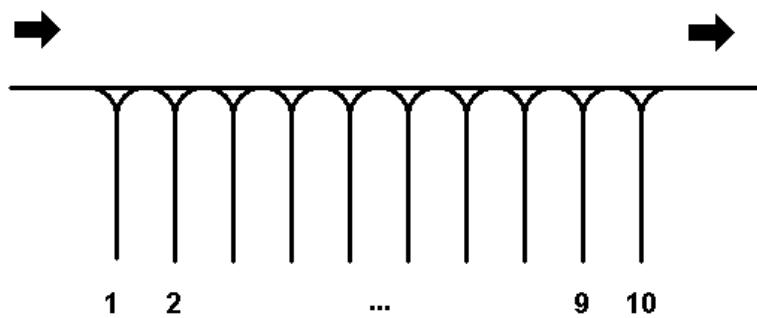
Раскраска “ходом коня” решает задачу. Формально, цвет клетки с координатами  $(i, j)$  можно положить равным  $(3i + j) \bmod 5 + 1$ .

3	4	5	1	2	3
1	2	3	4	5	1
4	5	1	2	3	4

## Задача D. Сортировка вагонов

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

На рисунке изображена схема сортировочной станции с 10 сортировочными ветками. На вход станции (слева) поступают  $n$  вагонов. Известен порядок, в котором они должны покинуть станцию. Кроме того, для упрощения работы станции на движение вагонов накладывается ограничение — они могут двигаться только слева направо (от веток с меньшим номером к веткам с большим номером, но не наоборот).



Ваша задача — по заданному порядку вагонов составить план сортировки. Сортировочные ветки достаточно длинны, чтобы вместить любое количество вагонов.

## Формат входного файла

Первая строка входного файла содержит единственное целое число  $n$  — количество вагонов ( $1 \leq n \leq 1000$ ). Вторая строка содержит  $n$  различных целых чисел  $a_1, a_2, \dots, a_n$  — номера вагонов, поступающих на вход сортировочной станции ( $1 \leq a_i \leq n$ ), в порядке поступления. Первым в списке находится номер вагона, находящегося ближе всего ко входу (а не самый левый)!

Номера вагонов определяют порядок, в котором они должны покинуть станцию. Вагон номер 1 должен выехать первым, номер  $n$  — последним.

## Формат выходного файла

Если сортировка невозможна, выведите ‘Impossible’ без кавычек.

Иначе выведите список команд, приводящий к сортировке вагонов. Каждая команда записывается в отдельной строке и имеет вид ‘in out’, где **in** и **out** — номер ветки, из которой перемещается вагон, и номер ветки в которую он перемещается (**in** строго меньше **out**). Для упрощения записи вход станции кодируется числом 0, а выход — числом 11.

В результате сортировки все вагоны должны оказаться на выходе в правильном порядке.

## Пример

<b>d.in</b>	<b>d.out</b>
4	0 1
4 2 1 3	0 2
	0 11
	2 11
	0 11
	1 11

## Разбор задачи D. Сортировка вагонов

Докажем следующий более общий факт: с помощью  $k$  сортировочных веток можно всегда отсортировать по крайней мере  $n = 2^k$  вагонов. Конкретные “круглые” ограничения  $k = 10, n \leq 1000$  были выбраны для того, чтобы решение не было таким очевидным.

Проведем рассуждение по индукции. База тривиальна: с помощью 0 сортировочных веток можно отсортировать 1 вагон.

Переход использует идею сортировки слиянием. Разобьем входящие вагоны на две группы по  $2^{k-1}$  вагонов в каждой, которые идут одна за другой. По предположению индукции, первую группу можно отсортировать в

любом порядке с помощью  $k - 1$  сортировочных линий. Отсортируем их в обратном порядке и загоним в последнюю,  $k$ -ю ветку. Таким образом, вагон с наименьшим среди этой группы номером сможет выехать из неё первым, с наибольшим — последним. Теперь первые  $k - 1$  веток освободились, и с помощью них мы можем отсортировать вторую группу вагонов — теперь уже в прямом порядке. Для простоты можно представить, что все они разместились в промежутке между  $k - 1$  и  $k$  ветками. Чтобы получить окончательный порядок, достаточно провести слияние двух упорядоченных последовательностей вагонов.

При практической реализации задачи вагоны второй группы после сортировки будут размещены в разных ветках от 1 до  $k - 1$ , т.к. формат вывода не предполагает использования промежутков между ветками. Придется формировать вторую последовательность “на лету” — слиянием этих  $k - 1$  упорядоченных последовательностей (с последующим слиянием с  $k$ -й).

### **Задача Е. Максимальный поток**

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

Дан неориентированный граф  $G$ , множество вершин которого  $V$  и множество рёбер —  $E$ . Кратные рёбра допускаются, петли — нет. Обозначим через  $n$  количество вершин графа, через  $m$  — количество рёбер. У каждого ребра  $(u, v) \in E$  определена пропускная способность  $C_{u,v}$ . Поток  $F$  из вершины  $s$  (исток) в  $t$  (сток) можно представить как поток вещества, которое могло бы пройти по сети от истока к стоку, если рассматривать граф как сеть труб с данными пропускными способностями. Необходимо выделить в графе любые две вершины  $s$  и  $t$  ( $s \neq t$ ), такие, что максимальный поток между ними равен сумме пропускных способностей всех ребер, инцидентных одной из этих вершин.

#### **Формат входного файла**

Первая строка входного файла содержит натуральное число  $n$  ( $2 \leq n \leq 10^5$ ) и  $m$  ( $0 \leq m \leq 10^5$ ). Следующие  $m$  строк содержат по три числа,  $u, v, c$  ( $1 \leq u, v \leq n$ ,  $1 \leq c \leq 10^9$ ), соответственно две вершины и пропускная способность. Все пропускные способности целые.

#### **Формат выходного файла**

В первой строке выведите пару чисел  $s$  и  $t$  (вершины удовлетворяющие

условию задачи), а также через пробел максимальный поток между ними. Если ответа не существует, выведите -1.

### Пример

e.in	e.out
4 4 1 2 2 1 3 3 2 4 3 3 4 1	1 3 4

### Разбор задачи Е. Максимальный поток

Алгоритм, предложенный автором этой задачи, является жадным и заключается в следующем. Изначально инициализируется пустое множество  $A$ . Далее, на каждой итерации выбирается некоторая вершина, расстояние которой от текущего множества  $A$  максимально и добавляется в это множество. Утверждается, что последняя и предпоследняя вершины, добавленные в множество  $A$ , будут удовлетворять условию задачи.

Данный алгоритм используется, как одна из фаз алгоритма Штора-Вагнера нахождения глобального минимального разреза, и был рассмотрен на лекции в полном объеме с доказательством.

Ограничения задачи требовали реализации данного алгоритма за  $O(e \log(v))$ , используя структуру данных, позволяющую извлекать максимум за  $O(\log(v))$ , где  $v$  — количество вершин,  $e$  — количество ребер.

### Задача F. Митохондриальная ДНК

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

Митохондриальная ДНК (мтДНК) — ДНК, локализованная (в отличие от ядерной ДНК) в митохондриях, органоидах эукариотических клеток.

---

Википедия

С точки зрения вычислительного биолога, основное отличие мтДНК от ядерной ДНК — циклическая структура первой. Это затрудняет её обработку, так как одна и та же мтДНК может быть представлена по-разному.

Ваша задача — написать программу, которая находила бы сходство между двумя неточно заданными мтДНК одинаковой длины. Неточно заданная мтДНК в этой задаче понимается как циклическая строка, состоящая из символов ‘A’, ‘C’, ‘G’, ‘T’ и ‘?’. Здесь четыре буквы кодируют четыре возможных основания, а знак вопроса — произвольное основание. Циклические строки одинаковой длины можно *выровнять* — циклически сдвинуть, получив две обычные строки — различными способами. Сходством полученных (обычных) строк называется количество совпадений оснований в одинаковых позициях строк, с учетом того, что вместо знака вопроса может быть подставлено любое основание. Сходством двух мтДНК называется максимально возможное сходство строк, полученных в результате некоторого выравнивания этих мтДНК.

### **Формат входного файла**

Входной файл содержит две строки. Каждая из строк задает одну мтДНК в формате, описанном в условии. Длина обеих строк одинакова и находится в пределах от 1 до 80 000 символов.

### **Формат выходного файла**

Выведите величину сходства заданных мтДНК.

### **Пример**

<b>f.in</b>	<b>f.out</b>
ACGT	4
GTAC	
CC	1
AC	
CCC?C	4
CCGCA	
ACGTACGT	7
?GTAC?TT	

### **Разбор задачи F. Митохондриальная ДНК**

Рассмотрим более простой вариант — пусть обе строки определены однозначно (символьные маски отсутствуют). Тогда сходство двух циклических

строк может быть сведено к 4 задачам о циклической корреляции

$$c_k = \sum_i s_i t_{i+k} \quad k = 0, 1, \dots, n-1,$$

где  $s_0 s_1 \dots s_{n-1}$  и  $t_0 t_1 \dots t_{n-1}$  — две последовательности нулей и единиц. Для этого достаточно рассмотреть каждую из 4-х букв алфавита оснований и заменить ее вхождение в строке на 1, а отсутствие — на 0.

Задача о циклической корреляции может быть сведена к задаче умножения полиномов (см. задачу А (“ДНК роботов”) дня Дмитрия Жукова). При выбранных ограничениях подходит любой более-менее быстрый способ умножения — алгоритм Карацубы за  $O(n^{1.59})$  или быстрое преобразование Фурье за  $O(n \log n)$ .

Осталось понять, как обрабатывать маски (знаки вопроса). Просто заменить каждое из их вхождений на 1 в обеих строках ошибочно — на тесте  $S = '?'$ ,  $T = '?'$  программа найдет 4 совпадения вместо 1. Чтобы исправить решение, можно искусственно внести в него ассиметрию — знаки вопроса в строке  $S$  заменять на 0, а в строке  $T$  — на 1. В результате мы найдем сходство всех известных символов строки  $S$  для всех сдвигов, с учетом масок строки  $T$ . Для получения окончательного ответа нужно прибавить к полученному числу количество масок в строке  $S$ , так как они всегда совпадут с любым символом.

### Задача G. Кратные запросы

Имя входного файла:	<code>g.in</code>
Имя выходного файла:	<code>g.out</code>
Ограничение по времени:	4 с
Ограничение по памяти:	256 Мб

Дана последовательность чисел  $\{a_1, a_2, \dots, a_n\}$ ,  $a_i = 0$ . На вход поступают запросы вида:

1. Изменить значение  $a_i$  на заданное.
2. Найти сумму  $a_i$ , где  $i$  кратно  $k$ .

### Формат входного файла

Первая строка входного файла содержит натуральное число  $n$  ( $1 \leq n \leq 10^6$ ) и  $q$  — количество запросов ( $1 \leq q \leq 10^5$ ). Следующие  $q$  строк содержат запросы. Первое число каждой строки — тип запроса (1 или 2). Далее, если это запрос вида 1, то следует два числа —  $i$  и  $v$

( $1 \leq i \leq n$ ,  $1 \leq v \leq 10^4$ ), соответственно номер элемента и новое значение. Если это запрос вида 2, то следует  $k$ ,  $1 \leq k \leq n$ . Все числа во входе целые.

### Формат выходного файла

Для каждого запроса второго типа выведите в файл отдельную строку с суммой элементов, номера которых кратны  $k$ .

### Пример

<b>g.in</b>	<b>g.out</b>
5 5	2
1 2 2	100
1 3 6	
2 2	
1 2 100	
2 2	

### Разбор задачи G. Кратные запросы

В данной задаче был задан массив элементов (изначально заполненный нулями) и требовалось быстро отвечать на запросы вида изменить значение некоторого элемента и посчитать сумму элементов, индексы которых кратны  $k$ .

Одно из решений подразумевало, что в каждый момент времени в отдельном массиве будут храниться актуальные суммы элементов, что обеспечит ответ на запросы второго типа за  $O(1)$ . Для этого необходимо было при изменении элемента массива с номером  $i$  обновить суммы элементов кратных  $k$ , где  $i$  делится на  $k$ .

Очевидно, что это можно сделать либо за  $O(p_i)$ , где  $p_i$  — количество делителей числа  $i$ , либо за  $\sqrt{i}$  обычным перебором, что тоже устраивало ограничения данной задачи.

### Задача H. Ориентирование

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

На заре развития авиации самолеты не были так хорошо оснащены, как сейчас. Компас был практически единственным навигационным прибором, и если он отказывал — летчику оставалось ориентироваться на местности

по солнцу и карте. К сожалению, пасмурная погода не редкость в нашем климате.

Пусть летчик четко идентифицировал три близлежащих ориентира  $A$ ,  $B$ ,  $C$ . Тогда он может измерить мгновенные курсовые углы этих ориентиров и попытаться восстановить координаты своего самолета (курсовый угол — это угол между направлением движения самолета и объектом, откладываемый по часовой стрелке).

Ваша задача — реализовать программу, которая делает то же самое автоматически.

### **Формат входного файла**

Первые три строки входного файла содержат по 2 вещественных числа  $x_i, y_i$  — координаты  $i$ -го ориентира ( $-10\ 000 \leq x_i, y_i \leq 10\ 000$ ).

Четвертая строка содержит 3 вещественных числа  $\varphi_1, \varphi_2, \varphi_3$  — курсовые углы трех ориентиров ( $0 \leq \varphi_i < 360$ ) в градусах.

Никакие три точки из четырех (три ориентира, самолет) не находятся на одной прямой. Единственность ответа гарантируется.

### **Формат выходного файла**

Выведите пару чисел — координаты самолета. Абсолютная или относительная погрешность каждой из координат не должна превышать  $10^{-9}$ .

### **Пример**

<b>h.in</b>	<b>h.out</b>
-10 0	0.0 0.00000000003
-5 5	
5 5	
275 320 50	

### **Разбор задачи Н. Ориентирование**

Известно, что геометрическое место точек, из которых фиксированный отрезок виден под некоторым ориентированным углом  $\varphi$  есть дуга окружности (за исключением случаев  $\varphi = 0$  и  $\varphi = \pi$ ).

По данным задачи мы можем вычислить углы, под которыми летчик видит отрезки  $AB$  и  $BC$ , и построить соответствующие окружности. Они обязательно имеют одну точку пересечения  $B$ , а также могут иметь еще одну точку пересечения или бесконечное их количество (если окружности совпадают).

Поскольку в задаче гарантировалась единственность решения, то случаи с одной или бесконечным числом общих точек невозможны. Поэтому

можно было не заботиться об учете нужных дуг — достаточно построить две окружности, найти две точки их пересечения и вывести ту, которая не совпадает с ориентиром.

### Задача I. Гонки

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

В городе X близится День города. Причём не просто в городе, а в столице страны У. Мэр города, который привык мыслить космическими масштабами, решил отметить такую дату по особому. Он решил перекрыть некоторые улицы и устроить на них гонки. А чтобы событие было ещё зрелищнее, мэр решил выбрать маршрут максимальной сложности. Сложность маршрута по его понятиям это сумма сложностей самого лёгкого и самого трудного участков. Маршрут должен быть циклическим. Так как карта города довольно большая, мэр поручил Вам найти самый сложный маршрут. К счастью, он подписал на каждой улице её сложность. Таким образом, карта города представляет собой неориентированный взвешенный граф с перекрёстками и улицами в качестве вершин и рёбер.

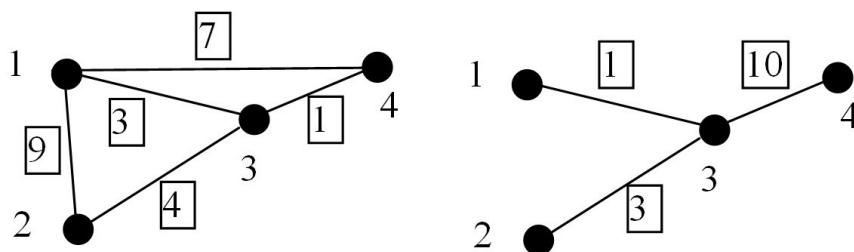


Рис. 1: Иллюстрация к тестам

### Формат входного файла

Первая строка входного файла содержит два целых числа  $n$  — количество перекрёстков в городе X и  $m$  — количество улиц ( $1 \leq n \leq 100\,000$ ,  $0 \leq m \leq 100\,000$ ). Далее идут  $m$  строк с описанием улиц. В каждой из этих строк записаны через пробел по три целых числа:  $x_i$ ,  $y_i$ ,  $w_i$ .  $x_i$ ,  $y_i$  — номер начального и конечного перекрёстков улицы ( $1 \leq x_i \leq n$ ,  $1 \leq y_i \leq n$ ,  $x_i \neq y_i$ ). Для каждой пары вершин может быть не больше одного ребра.  $w_i$  — сложность улицы ( $1 \leq w_i \leq 1\,000\,000$ ).

## Формат выходного файла

В выходной файл выведите в единственной строке единственное число — наибольшую сложность кругового маршрута или 0, если невозможно выбрать ни один маршрут.

### Пример

i.in	i.out
4 5 1 4 7 1 3 3 1 2 9 3 4 1 3 2 4	12
4 3 3 1 1 3 4 10 3 2 3	0

## Разбор задачи I. Гонки

В этой задаче требовалось найти во взвешенном неориентированном графе такой цикл, у которого величина сложности была бы максимальна. Сложность цикла это сумма значений минимального и максимального рёбер на нём. Для решения можно применить метод бинарного поиска по ответу. Таким образом, нам необходимо лишь уметь проверять можно ли найти цикл сложности заданной константы  $C$  или больше. Отнимем от величины всех рёбер  $\frac{C}{2}$ . Тогда задача состоит в том, чтобы найти цикл со сложностью большей либо равной нулю. Для этого применим следующий алгоритм. Рассмотрим наименьшее неотрицательное ребро. Пусть его стоимость  $D$ . Добавим в систему непересекающихся множеств все отрицательные рёбра, у которых стоимость больше либо равна  $-D$ . После этого добавим ребро стоимости  $D$ . Если при добавлении этого ребра образовался цикл, то понятно, что он и есть искомый. Если же цикл не появился, то берём следующее по величине не отрицательное ребро после  $D$ . Пусть его стоимость  $D_1$ . Добавим в систему непересекающихся множеств все рёбра стоимостью от  $-D - 1$  до  $-D_1$  включительно. Потом добавим само ребро стоимости  $D_1$ . Если при добавлении этого ребра образовался цикл, то он и есть искомый. Так продолжаем, пока не закончатся все положительные рёбра. В таком случае искомого цикла не существует.

Можно оценить сверху время работы данного алгоритма:  $O(E \log C f^*(V))$ , где  $C$  – максимальная стоимость ребер,  $f^*(V)$  – обратная функция Аккермана.

### Задача J. TR3N

Имя входного файла: j.in

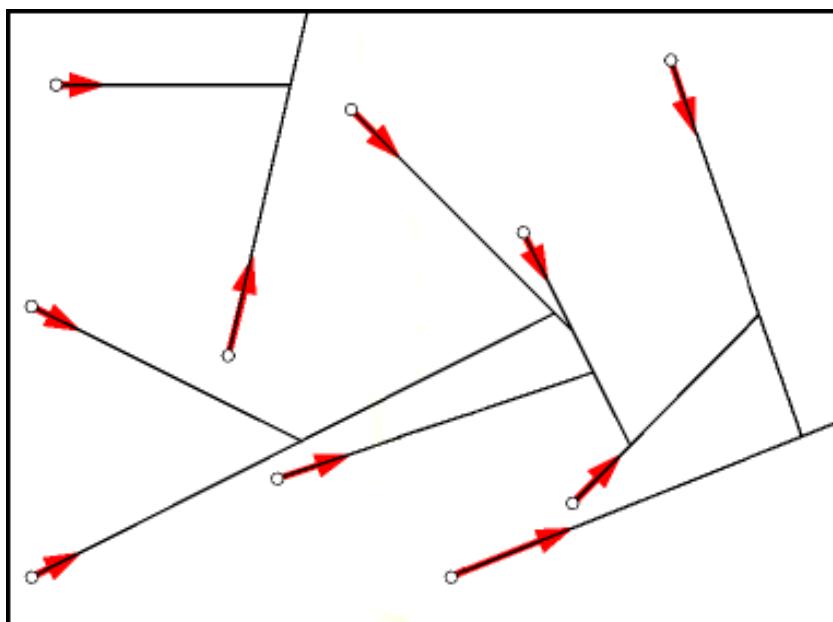
Имя выходного файла: j.out

Ограничение по времени: 10 с

Ограничение по памяти: 256 Мб

Режиссеры планирующегося триквела решили, что неуспех продолжения TRON связан с недостаточным количеством спецэффектов. Были рассмотрены различные способы улучшить данный аспект фильма. В частности, предлагалось снять сцену соревнования световых мотоциклов *очень многих игроков одновременно*.

Введем на плоскости прямоугольную систему координат. Соревнование проходит в комнате, ограниченной непроницаемыми стенами, заданными уравнениями  $x = \pm W$ ,  $y = \pm H$ . В игре участвует  $n$  мотоциклов.  $i$ -й игрок начинает движение в точке с координатами  $x^i, y^i$ . По команде все мотоциклы начинают двигаться (возможно, с разной скоростью), оставляя непроницаемую стену вдоль своей траектории движения. Игрок, врезавшийся в стену, погибает.



Пока что сценарий этого эпизода не проработан до конца, так что для от-

ладки компьютерной графики используется следующая простая модель — все игроки движутся прямолинейно с постоянной скоростью. Известны проекции скорости  $i$ -го мотоцикла на координатные оси  $v_x^i, v_y^i$ . Кроме того, в результате случайного стечения обстоятельств все мотоциклисты двигаются в сторону возрастания абсцисс (формальное определение этому дано в описании входного файла).

Прорисовка одного метра оставленной световым мотоциклом стены требует 1 мегафлопс ресурсов. Определите суммарный объем вычислений, требуемых для прорисовки всей сцены (внешние стены и мотоциклы не прорисовываются).

### Формат входного файла

В первой строке содержится три целых числа  $n, W, H$  — количество мотоциклов и размеры комнаты ( $1 \leq n \leq 100\,000, 1 \leq W, H \leq 100\,000$ ).

Следующие  $n$  строк содержат по четыре вещественных числа  $x^i, y^i, v_x^i, v_y^i$  — координаты начальных позиций и скорости мотоциклов ( $-W < x^i < W, -H < y^i < H, 0 < v_x^i \leq 10, -10 \leq v_y^i \leq 10$ ).

Гарантируется, что в процессе движения никакие два мотоцикла не столкнутся друг с другом. Также во входных файлах отсутствуют случаи, когда один мотоцикл наталкивается на точку старта другого.

### Формат выходного файла

Выполните единственно число — суммарный объем вычислений, требуемых для прорисовки всей сцены, в гигафлопсах. Абсолютная или относительная погрешность ответа не должна превышать  $10^{-9}$ . Один гигафлопс равен 1000 мегафлопсам.

### Пример

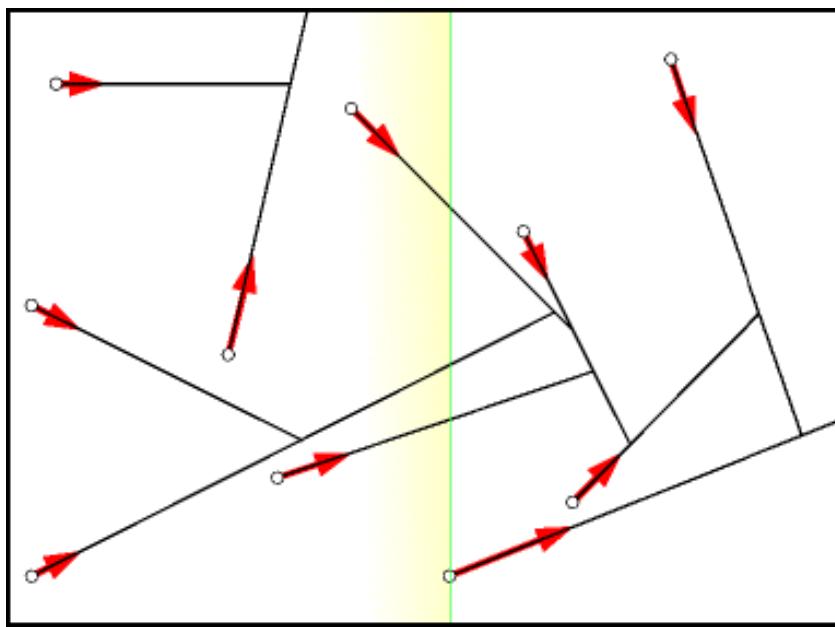
j.in	j.out
2 1000 1000 0 0 1 0 0 -200 0.5 0.5	1.28284271247462
2 1000 1000 0 0 1 0 0 -200 2 2	1.61421356237309

### Разбор задачи J. TR3N

Тривиальное решение за  $O(n^2 \log n)$  состоит в том, чтобы для каждой пары мотоциклов найти их момент пересечения и отсортировать эти события по времени прихода более позднего мотоцикла в точку пересечения. К

сожалению, при предложенных ограничениях такое решение не пройдет по времени.

Граф пересечений можно построить за  $O(n \log n)$ , используя алгоритм заметания. Будем заметать плоскость вертикальной прямой слева направо, храня текущее пересечение прямой с графом в сбалансированном двоичном дереве. Количество точек в пересечении может изменяться двумя способами: когда прямая проходит стартовую точку мотоцикла (“рождение”), и когда прямая проходит точку пересечения траекторий двух смежных мотоциклов (“смерть”). Эти события можно хранить в очереди по приоритетам, упорядоченными по абсциссе.



Вначале очередь содержит только  $n$  рождений и  $n$  смертей в результате столкновения со стенками. Чтобы обработать рождение, вставим новый мотоцикл в двоичное дерево, глобально отметим его „живым” и добавим в очередь два новых события смерти — результат пересечения его траектории с траекториями соседей в дереве. Чтобы обработать смерть, мы проверяем что оба участвующих в событии мотоцикла живы — иначе это событие игнорируется. Затем мы определяем, какой мотоцикл пришел в точку встречи позже, удаляем его из двоичного дерева, отмечаем „мертвым” и прибавляем пройденный им путь к ответу. Кроме того, необходимо добавить новое событие смерти, вызванное пересечением траекторий двух его соседей в дереве.

Всего будет обработано не более  $3n$  событий, и каждое событие обрабатывается за время  $O(\log n)$ .

## Задача К. Хамелеоны

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

В стране Серобуромалинии живет  $a$  серых,  $b$  бурых и  $c$  малиновых хамелеонов. Когда встречаются два хамелеона разного цвета, они одновременно перекрашиваются в третий цвет (например, серый и бурый становятся малиновыми). Может ли так случиться, что через некоторое время все хамелеоны будут одного цвета? Если да, то какого?

### Формат входного файла

Первая строка входного файла содержит три целых числа  $a, b, c$  — начальное количество серых, бурых и малиновых хамелеонов ( $1 \leq a, b, c \leq 1500$ ).

### Формат выходного файла

Если ситуация, когда все хамелеоны окрашены в один цвет, невозможна — выведите ‘Impossible’.

Если возможна одновременная покраска хамелеонов в серый цвет (и ни в какой другой) — выведите ‘Grey’. Аналогично, выведите ‘Brown’ или ‘Crimson’ в случае бурого или малинового цветов.

Если хамелеоны могут одновременно покрасится в один цвет разными способами (скажем, в серый или в бурый) — выведите ‘Ambiguous’.

### Пример

<b>k.in</b>	<b>k.out</b>
1 1 0	Crimson
1 1 1	Ambiguous
11 0 1	Impossible

## Разбор задачи К. Хамелеоны

Задача имеет 2 решения — математическое и программистское. Приведем оба.

Можно заметить, что величины  $(a - b) \pmod{3}$ ,  $(a - c) \pmod{3}$  и  $(b - c) \pmod{3}$  являются инвариантами (не меняются в процессе перекрасок). Это можно использовать для доказательства *невозможности* одновременной перекраски в какой-то цвет. Например, при полной перекраске в серый

цвет эти три числа в итоге станут равны  $(a + b + c) \pmod{3}$ ,  $(a + b + c) \pmod{3}$  и 0, соответственно. Если это условие нарушено в самом начале, то перекраска очевидно невозможна.

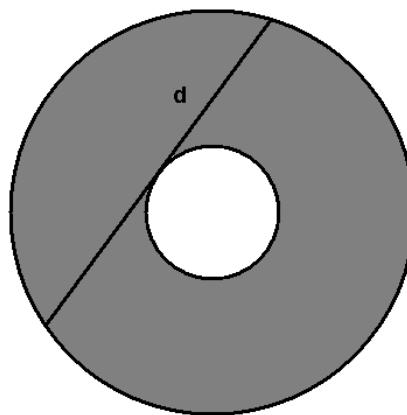
Справедливо и обратное утверждение — если введенные инварианты начального и конечного состояний совпадают, то эти состояния достижимы друг из друга с помощью определенной последовательности перекрасок. Поэтому достаточно рассмотреть остатки от деления  $a, b, c$  на 3 и задача сводится к перебору конечного числа случаев за  $O(1)$ .

Альтернативное решение предполагало обход графа допустимых состояний с помощью поиска в ширину или поиска в глубину (первый способ предпочтительнее из-за возможных проблем со стеком). Поскольку одно состояние кодируется двумя числами  $a$  и  $b$ , то всего доступно  $O((a+b+c)^2)$  состояний. При указанных ограничениях такое решение тоже укладывалось в предоставленный лимит времени.

### Задача L. Площадь кольца

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

На плоскости заданы две концентрические окружности. У внешней окружности проведена хорда, одновременно являющаяся касательной к внутренней окружности. Найдите площадь кольца, ограниченного окружностями (заштриховано на рисунке).



### Формат входного файла

Первая строка входного файла единственное вещественное число  $d$  —

длина хорды ( $0 \leq d \leq 100$ ).

## Формат выходного файла

Выведите искомую площадь. Абсолютная или относительная погрешность ответа не должна превышать  $10^{-9}$ .

### Пример

1.in	1.out
3.56824823230554	10.0

## Разбор задачи L. Площадь кольца

Эта задача тоже имеет 2 решения — геометрическое и физическое. Второе более интересно, и больше нравится автору.

Встретив задачу на олимпиаде по программированию, можно смело предположить что требуемый ответ является функцией от входных данных (иначе задача попросту некорректна). Таким образом, в нашем случае нужно найти функцию  $S(d)$  — площади кольца от длины хорды.

Заметим, что единственный входной параметр  $d$  имеет размерность длины (например, это могут быть метры), а ответ — размерность площади (соответственно, метры квадратные). Таким образом из соображений размерности немедленно следует, что  $S(d) = \alpha d^2$ , где  $\alpha$  — безразмерная константа.

Первый способ найти  $\alpha$  состоит в подстановке чисел из примера входного и выходного файлов. Таким образом можно получить значение  $\alpha \approx 0.78539816339744929088916003655523$ , чего вполне достаточно для успешного решения задачи.

Тем не менее, вбивать 15 знаков в калькулятор бывает лень. Можно обойтись без этого и найти константу точно. Для этого положим радиус внутренней окружности равным 0, тогда кольцо выродится в больший круг. Его площадь будет равна  $\pi d^2 / 4$ , следовательно  $\alpha = \pi / 4$ .

## День пятый (16.02.2011 г.) Конкурс Копелиовича Сергея Владимировича

### Об авторе...

**Копелиович Сергей Владимирович** родился в Санкт-Петербурге в 1989 году. Закончил Физико-математический лицей № 30. В 4-6 классах занимался в кружке по математике. Олимпиадами по программированию начал активно заниматься в 9-м классе.

Студент 5-го курса Санкт-Петербургского государственного университета. Член научного комитета всероссийской школьной олимпиады по информатике и сборов по подготовке к международной олимпиаде школьников.

В ACM соревнованиях выступал в команде Burunduchki.

С 2008-2009 учебного года регулярно читает лекции в СПбГУ. Ссылка на планы лекций <http://kruzhok.spbgu.ru/09e>

С 2010-го года стажер в санкт-петербургском офисе Яндекса.



Основные достижения:

- Золотые медали на IOI в 2005 (Польша) и 2006 (Мексика) годах.
- Бронзовая и золотая медали в 2008 (Канада) и 2009 (Швеция) годах.

### Теоретический материал. Перебор с отсечениями

#### 1 Persistent Data Structures

Для понимание этой части очень желательно знакомство с деревьями отрезков.

## 1.1 Введение

Для начал поймем, что такое **Persistent**. Пусть у нас есть массив  $a_0$  и мы его меняем, пишем  $a_0[i] := x$ , старый массив не меняется, создается новый массив  $a_1$ . Если мы пишем теперь  $a_0[j] := y$ , создается новый массив  $a_2$ , где в позиции  $i$  нет  $x$ -а.

Получается дерево версий. Старые версии структур живут вечно, мы их никогда не меняем. Мы только создаем новые структуры, новые версии.

Сейчас мы научимся делать любую структуру данных **Persistent**-ной. При этом память и время станут больше в  $O(\log n)$  раз. Для бинарных деревьев увеличится только память.

## 1.2 Массив = Дерево отрезков, Дерево - это просто

Структуры данных в основном состоят из массивов и бинарных деревьев.

Для начала представим массив в виде дерева отрезков. Дерево мы будем хранить не в другом массиве, а как произвольное бинарное дерево со ссылками на левое и правое поддеревья. Обращение к одному элементу массива (и чтение, и запись) теперь работают не за  $O(1)$ , а за  $O(\log N)$ .

Вернемся к деревьям. Как дерево сделать *Persistent*-ным? Рассмотрим следующий код, надеюсь, он полностью ответит на поставленный вопрос:

```
tree Add( tree t, int x )
{
    if (t == null)
        return new tree(null, null, x);
    if (x < t.x)
        return new tree(Add(t.left, x), t.right, t.x);
    else
        return new tree(t.left, Add(t.right, x), t.x);
}
```

Обратите внимание, старое дерево мы ни разу не поменяли. Подведем итог:

*С этого момента весь мир для нас — деревья, а деревья мы умеем делать **Persistent**. Каждая версия структуры данных — корень дерева. Мы храним массив версий, массив корней. Построенная нами конструкция — это несколько деревьев. Вместе они образуют ациклический граф. Одно дерево однозначно задается корнем.*

## 1.3 Garbage Collection

Если нам нужны все версии, все их нужно хранить. После  $k$  запросов, памяти будет использовано  $O(k \log k)$ .

Представим другую ситуацию: только 3-4 версии все еще нужны, память из под остальных можно очистить. В общем случае, если нужных версий в каждый момент времени  $O(1)$ , память можно сократить до  $O(k)$  в каждый момент времени.

Способ 1:

Будем хранить число ссылок на каждую вершину. Если какая-то вершина является корнем, число ссылок на нее тоже нужно увеличить на единицу.

Если какая-то версия нам уже не нужна, уменьшаем число ссылок на корень этой версии, если число ссылок стало нулем, эта вершина нам больше не нужна. Удалим ее, освободим память. Удалять нужно рекурсивно, удаление вершины уменьшает число ссылок на другие вершины.

Способ 2:

Будем иногда (когда памяти осталось совсем мало и уже пора что-то делать) делать так: берем все живые корни и обходим их деревья, помечая, что все эти вершины живые. После этого очищаем память из-под всех вершин, не помеченных, как живые.

## 1.4 Демонстрация мощи слова **Persistent**. Решим задачку.

Многие задачи имеют Offline решение гораздо более простое, чем Online. Persistent структуры позволяют нам многие Offline решения обобщить на Online случай.

Сделаем это на примере задачи *Даны  $n$  точек на плоскости, запрос = посчитать число точек в прямоугольнике*. Стороны прямоугольника, конечно, параллельны осям координат.

Offline решение этой задачи — одномерное дерево отрезков со сканирующей прямой. Пройдемся сканирующей прямой по точкам. Все деревья отрезков сделаем Persistent и сохраним в памяти. Произошло чудо, теперь, когда нам приходит очередной прямоугольник, оба дерева отрезков, которые нужны для ответа на запрос уже посчитаны, к ним можно обратиться.

Добавлять новые точки мы так и не научились, но, если набор точек фиксирован, на запросы мы умеем отвечать Online.

## 1.5 Заключение

1. Многие структуры данных состоят из массивов и деревьев.
2. Дерево естественным образом делается персистентным (без потери времени, памяти в  $\log N$  раз больше).
3. Массив — это тоже дерево. Дерево отрезков.

## 2 Перебор!

Для понимания этой части обязательно знакомство с рекурсией.

### 2.1 Читерские методы

#### 2.1.1 Предподсчет

Представьте такую задачу: вам дан бинарный файл. Вы знаете, что это программа, которая для  $n$  от 1 до 500 умеет искать число кактусов из  $n$  вершин с точностью до изоморфизма. Эту программу написали участники конテスト. Вы не знаете, правильно ли программа работает, какие ошибки в ней сделаны, у вас нет исходного кода программы.

Ваша задача — написать свою программу, работающую так же, как и данная вам на всех тестах. Решение этой задачи — запустить данную вам программу на всех тестах от 1 до 500 и отправить предподсчитанный массив.

Это первое читерство. Имя его — **предподсчет**. Не забывайте про него.

#### 2.1.2 Ленивое DP = перебор + запоминание

Чтобы написать динамику нужно написать перебор и добавить в него запоминание. Можно воспринимать это как ленивую динамику, можно — как оптимизацию к перебору. Часто, если не получается написать простую динамику по профилю, можно написать рекурсивный перебор и добавить в него отсечение “динамика по профилю”.

Для тех, кто плохо знаком с ленивой динамикой: рассмотрим задачу *найти число путей из  $s$  в  $t$  в ациклическом графе*. Будем для каждой вершины  $v$  считать  $f(v)$  — число путей из  $v$  в  $t$ .  $f(v) = \sum_{x \text{ = дитё } v} f(x)$ .

Запускаем  $dfs$  из вершины  $s$  и рекурсивно считаем функцию  $f$  для всех вершин графа. Решение работает за  $O(E)$ . Чем это решение отличается от перебора всех путей из  $s$  в  $t$ ? Ответ:

```
if (used[v])
    return f[v];
used[v] = 1
```

Благодаря этому `if`-у перебор превращается в динамику.

#### 2.1.3 Жадность и перебор

Любую жадность можно улучшить до перебора. Жадность работает на некотором графе состояний. При этом в каждый момент времени выбирается максимальное по некоторой функции ребро. Можно вместо выбора

максимума сделать сортировку и запустить перебор по всем ребрам начиная с максимального. Работать будет не хуже. А на маленьких тестах даже лучше.

**Пример:** решаем задачу о рюкзаке (набрать в рюкзак ограниченного веса вещи так суммарной стоимостью) жадностью “брать каждый раз вещь с max удельной стоимостью”, чтобы превратить жадность в перебор, будем сортировать вещи по этому критерию и реализуем идею рекурсивно.

### **2.1.4 Ленивость перебора**

Не перебирайте слишком много. Перебирайте значение объектов только, когда эти значения вам понадобились (для сравнения в `if`-е, например). Действуйте максимально лениво.

## **2.2 Постановка общей задачи**

Давайте разберемся, что же такое перебор, какую задачу он решает, и чему будет посвящена эта часть лекции. “Перебором” мы будем решать 4 задачи на графе состояний.

- Проверка достижимости “хорошей” вершины.
- Поиск невзвешенного кратчайшего пути.
- Поиск взвешенного пути  $\min$  веса.
- Поиск взвешенного пути  $\max$  веса.

Последняя задача сводится к предыдущей домножением весов на  $-1$ , поэтому будем считать, что задач всего 3.

Поговорим о нашем графе. Граф, на котором мы решаем описанные 3 задачи задан неявно. Он всегда настолько большой, что сохранить в памяти его не получается. Иначе мы бы просто написали соответственно `dfs`, `bfs` и `Ford-Bellman` или `Dijkstra`.

Многие из предложенных решений будут, как и положено “перебором”, рекурсивными, но не все. Поговорим о рекурсии. Рекурсия порождает дерево рекурсии. Некоторые состояния этого дерева хочется склеить, так как это одна и та же вершина графа состояний. Рекурсивный же перебор — попытка написать алгоритмы `dfs`, `bfs` и `Ford-Bellman` рекурсивно. В отличии от `dfs`-а, `bfs` и `FB` написать рекурсивно не просто.

## **2.3 Меморизация (запоминание, хэширование состояний)**

### **2.3.1 `dfs`**

Для `dfs`-а это хэш-таблица. Собственно мы получаем ленившую динамику.

### 2.3.2 bfs

Для bfs-а, если мы его пишем нерекурсивно, все понятно — состояние добавляется в очередь только, если его еще нет в хэш-таблице. Если же bfs реализуется через рекурсию, можно проверять, улучшилось расстояние до вершины, или нет. Углублять рекурсию имеет смысл только если расстояние до вершины улучшилось. Время работы такого алгоритма в худшем случае  $O(VE)$  (т.к. для каждой вершины будет не более  $V$  улучшений).

### 2.3.3 FB

Рекурсивная реализация ничем не отличается от “bfs”-а (см. выше)

## 2.4 Метод ветвей и границ для dfs-а

Рассмотрим теперь так называемый метод ветвей и границ на примере двух задач — поиск Гамильтонова пути в графе и поиск максимального по длине пути в невзвешенном графе.

Суть метода ветвей и границ в том, что не все ветви рекурсии нужно обходить, некоторые можно отсекать. Поводом для отсечения ветвей являются границы (оценки на то, насколько это поддерево интересно или неинтересно).

Для решения задачи прежде всего определим состояние: множество уже посещенных вершин и последняя вершина в пути. Состояний всего  $O(2^n n)$ , ребер  $O(2^n n^2)$ . У нас появился новый граф, граф состояний. Используем меморизацию (запоминание). В обоих задачах для каждого состояния нам важно только были мы уже в таком, или еще нет. Итого наше переборное решение работает за  $O(2^n n^2)$ .

Теперь новое, список отсечений:

1. **Отсечение по времени:** добавить его — никогда не лишнее.
2. **Отсечение по ответу:** это самое главное отсечение.  $CurrentValue + F(State) \geq Answer$ ,  $F(State)$  = количество достижимых вершин из текущего конца пути. Если мы ищем Гамильтонов путь,  $Answer$  изначально равен  $V - 1$ , иначе 0.
3. **Жадность, сортировка ребер:** давайте перебирать ребра не в произвольном, а в каком-то “хорошем” порядке. Один из вариантов такого порядка — сортировка по  $F(State)$ . Для задачи про Гамильтонов цикл хорошо известна лучшая жадность — порядок возрастания степени вершин. Интересный факт: задача про обход конем шахматной доски  $N \times N$  для  $N \leq 300$  решается без перебора, жадно. Для этого достаточно начать из случайной клетки доски, а жадность улучшить такой эвристикой: если степени равны, пытаемся максимально прижаться к краю доски.

4. **Перебор только  $k$  лучших ребер:** это первая из рассмотренных нами оптимизация (отсечение по времени не в счет), которая может привести к вердикту *Wrong Answer*. Тем не менее, мощь оптимизации велика. Мы только пока не знаем, какое  $k$  лучше взять.
5. **Iterative Increasing of  $k$ :** будем перебирать  $k$  во внешнем цикле  
`for (k=1; ; k++)`

Это попытка дать общую схему, которую легко обобщить на похожие задачи. Конкретно про Гамильтонов цикл можно добавить следующее: помогает поиск мостов за  $O(E)$  на каждом шаге рекурсии. Про путь *max* длины добавлю: для получения впечатляющих результатов достаточно первых двух из перечисленных оптимизаций.

**Важно:** Как работает отсечение по ответу? Нужно иметь функцию оценки  $F(State)$ : если  $CurrentValue + F(State) \geq Answer$ , можно оборвать рекурсию. Кроме этого нужно собственно иметь хорошее приближение ответа —  $Answer$ . Поиск в глубину сперва работает как жадность. Доходит до самого низа, получается оценка на ответ — текущий  $Answer$ .

## 2.5 dfs может быть не хуже bfs-а

Зададимся вопросом: мы ищем кратчайший путь, как его можно найти `dfs`-ом, и чем `dfs` может оказаться лучше нерекурсивного `bfs`-а с очередью? Главное преимущество `dfs`-а — отсечение по ответу. Недостаток — то, что в каждую вершину мы вынуждены заходить несколько раз (каждый раз при этом расстояние до вершины улучшается).

Решение проблемы = `Iterative Deepening`. Давайте дадим `dfs`-у ограничение по глубине `MAX_DEPTH`. И будем перебирать эту константу внешним циклом `for (MAX_DEPTH=1; ; MAX_DEPTH++)`. Это может ухудшить отсечение по ответу (иногда отсечение по ответу хорошо работает, только если мы позволяем `dfs`-у доходить до глубоких вершин графа). Мы получили новый алгоритм: `dfs` + `Iterative Deepening`.

Объясним, почему новый алгоритм по времени работы не хуже `bfs`-а. Казалось бы, мы должны каждый раз, для каждого нового `MAX_DEPTH` проделывать ту же работу, что и на предыдущей итерации, а `bfs` все делает только один раз. Заметим, что рост числа состояний экспоненциален от глубины. Т.е. `bfs` работает за  $Num(Ans)$ , а `dfs` + `Iterative Deepening` примерно за  $Num(Ans) + Num(Ans)/2 + Num(Ans)/4 + \dots$ , что асимптотически не превосходит  $Num(Ans)$ .

## 2.6 bfs = dfs = dfs + Iterative Deepening

`dfs` + `Iterative Deepening` лучше `bfs`-а тем, что в нем есть отсечение по ответу. Давайте в `bfs` добавим такое же отсечение. И в `dfs` + `Iterative Deepening`, и в `bfs`, и в `dfs` отсечение по ответу работает

по-разному, т.к. вершины перебираются в разном порядке, чтобы это ни-чем не портило первые 2 алгоритма, давайте сперва для улучшения оценки ответа запустим `dfs` на 0.5 секунд, а потом уже собственно алгоритм.

### **2.7 Улучшенный bfs, мажорирующий все другие методы**

Давайте подведем итог всем отсечениям, которые мы узнали и добавим еще одно, последнее. Получится отличный алгоритм.

1. Сперва запускаем `dfs` без `Iterative Deepening` на 0.5 секунд.  
Используем все оптимизации для `bfs`-а, которые мы проходили выше.
2. Теперь пишем `bfs`. Нерекурсивный. Ограничиваем `bfs`-у размер очереди.
3. Внутри `bfs`-а не забываем использовать отсечение по ответу. Это почти самое важное.
4. В очереди в каждый момент времени оставляем только лучшие состояния (жадность). Можно использовать кучу (очередь с приоритетами), можно, как только очередь переполнится в 2 раза, сортировать все состояние и оставлять нужное число лучших.
5. `Iterative Deepening` по `max` размеру очереди. Размер очереди каждый раз удваивается.
6. Отсечение по времени. Оно нам поможет :-)

Замечание: 5-й оптимайз почти мажорирует 1-й, если размер очереди начинать перебирать с единицы.

### **2.8 AB-отсечение**

Про него я сегодня рассказывать ничего не буду :-)

## **3 Метод откатов**

Для понимания этой части очень желательно знание СНМ (Системы Непересекающихся Множеств).

Сейчас мы решим несколько задач. Задачи будут усложняться. Все задачи имеют почти одно и то же условие *Дан граф, он как-то меняется. После каждого изменения нужно говорить, сколько компонент связности в графе.*

### **3.1 Задача 1: ребра добавляются**

СНМ в явном виде. Решение работает в `Online`.

### **3.2 Задача 2: ребра удаляются**

Сперва удалим все ребра, а потом будем их добавлять в обратном порядке. Решение работает только в `Offline`.

### **3.3 Задача 3: ребра сперва добавляются, а потом удаляются**

Разбиваем задачу на 2 части, решаем их отдельно. Решение работает только в Offline.

### **3.4 Задача 4: удалять можно только ребро добавленное последним**

Есть 2 решения.

Решение 1: Persistent :-)

Решение 2: Нам нужна только одна версия, текущая.

Память меняется. Мы меняем какие-то ячейки памяти, но мы помним, какие. Можно все изменения значений в памяти хранить в стэке (храним ячейку и старое значение) и откатывать по необходимости состояние памяти.

Асимптотика СНМ при этом портится до  $O(\log N)$ , т.к. нам важно, сколько 1 запрос работает в худшем случае в реальном времени, без амортизации.

Давайте подчеркнем главную идею: удаление ребра мы не делаем, мы читерим, откатываемся к предыдущему состоянию, в котором уже все подсчитано.

Решение полностью Online.

### **3.5 Задача 5: удалять можно произвольное существующее ребро**

Собственно об этом я и хотел рассказать. Решение предыдущей задачи можно назвать стеком. Решением этой задачи будет дерево отрезков. Идея решения не изменится — важно не делать Del, а откатываться назад.

Каждому запросу Add соответствует парный ему Del. И наоборот, каждому Del соответствует парный ему Add.

Построим на запросах дерево отрезков. Пусть мы хотим знать число компонент связности после всех запросов с номерами  $[L..R]$ . Какие-то Add-ы уже точно можно сделать, какие-то точно известно, что отменятся парным Del-ом. А про какие-то Add-ы мы пока не можем сказать ничего определенного. Храним все такие “неопределенные” Add-ы.

При рекурсивном спуске по дереву отрезков будем некоторые “неопределенные” Add-ы делать определенными. Эффекта два — мы больше не тащим их вглубь рекурсии, и нужно как-то поменять граф.

После рекурсивного спуска в конце концов следует рекурсивный подъем. При подъеме нужно отменить все изменения, которые мы сделали при спуске. Тут есть, как и в прошлой задаче, 2 пути — Persistent и хранить стэк изменений памяти.

Асимптотика решения ухудшилась уже до  $O(k \log^2 k)$ . Тем не менее, такому времени для настолько сложной задачи можно порадоваться. Заметьте, решение работает только в Offline.

## 4 Разбор как продолжение лекции

Данная лекция задумывалась таким образом, что на разбор задач останется какой-то теоретический материал. Например, на разборе будут рассказаны быстрые решения  $NP$ -задач “число подклик в графе” и “минимальное вершинное покрытие произвольного графа”.

### Задачи и разборы

#### Задача А. Карточки

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

На день рождения Вася подарил своему брату Пете набор из  $N$  карточек. Затем Вася и Петя написали по одному числу на каждой из карточек, Вася с одной стороны, а Петя — с другой. После этого братья разбросали карточки по комнате и убежали. Все это безобразие увидела их бабушка Людмила Петровна. Она много лет работала бухгалтером, поэтому она решила вспомнить молодость и придумала себе игру: нужно некоторые карточки перевернуть “васиной” стороной, а некоторые “петиной”, причем так, чтобы сумма была максимально возможной. Но, так как она любит обоих братьев одинаково сильно, она добавила дополнительное условие, что количество карточек, перевернутых “петиной” стороной, должно отличаться от количества карточек, перевернутых “васиной” стороной, не более, чем на  $K$ .

Людмила Петровна легко справилась с задачей, а Вы сможете?

#### Формат входного файла

В первой строке содержатся числа  $N$  и  $K$  ( $1 \leq N \leq 100\,000$ ,  $1 \leq K \leq N$ ) — количество карточек. Далее в  $N$  строках содержится описание карточек — два целых числа: первое число написано Васей на одной стороне карточки, второе написано Петей на другой стороне. Вася и Петя не знают чисел, которые по модулю превосходят  $10^9$ .

#### Формат выходного файла

В первой строке выходного файла должно содержаться одно число — максимально возможная сумма. Далее выведите  $N$  чисел — для каждой карточки выведите 1, если она Петиной стороной, и 0, если Васиной.

Если ответов несколько, выведите любой.

## Примеры

<b>a.in</b>	<b>a.out</b>
5 1	8
1 2	1 1 1 0 0
1 2	
1 2	
1 2	
1 2	
1 2	

## Разбор задачи А. Карточки

Изначально перевернем все карточки одной стороной. Теперь нам нужно перевернуть от  $\min$  до  $N - \min$  карт, где  $\min = \frac{n-k}{2}$ . Переворачивая  $i$ -ю карточку, мы увеличиваем ответ на  $b_i - a_i$ . Давайте отсортируем значения  $b_i - a_i$ . Максимальные  $\min$  штук точно возьмем. Минимальные  $\min$  штук точно не возьмем, оставшиеся можно брать, можно не брать, возьмем только положительные.

Асимптотика этого решения  $O(N \log N)$ . Чтобы избавиться от логарифма, можно вместо сортировки искать  $k$ -ю порядковую статистику за линейное время.

## Задача В. Функция

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

Вычислите функцию:  $f(n) = \begin{cases} 1 & \text{если } n \leq 2 \\ f(\lfloor 6 * n / 7 \rfloor) + f(\lfloor 2 * n / 3 \rfloor) & \text{если } n \bmod 2 = 1 \\ f(n - 1) + f(n - 3) & \text{если } n \bmod 2 = 0 \end{cases}$

## Формат входного файла

Входные данные содержат натуральное число  $n$  ( $1 \leq n \leq 10^{12}$ ).

## Формат выходного файла

Выведите значение функции по модулю  $2^{32}$ .

## Пример

<b>b.in</b>	<b>b.out</b>
7	10

## Разбор задачи В. Функция

Пишем рекурсивную функцию  $f(n)$ , добавляем меморизацию (для каждого  $n$  считаем функцию ровно один раз). Различных состояний мало.

P.S. Предполагалось, что после лекции решение будет очевидно.

## Задача С. Пути на доске

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

Рассмотрим бесконечную клетчатую доску.

Назовём *путём* из одной клетки в другую последовательность клеток, в которой каждые две идущие подряд клетки являются соседними по стороне. Длина пути — это количество клеток в нём, не считая начальную.

Назовём путь *простым*, если в нём не встречается двух одинаковых клеток.

Зафиксируем какую-то клетку на доске. Сколько существует простых путей заданной длины, начинающихся в этой клетке?

### Формат входного файла

В первой строке входного файла задано целое число  $n$  ( $0 \leq n \leq 22$ ).

### Формат выходного файла

В первой строке выходного файла выведите одно число — количество путей длины  $n$  из этой клетки.

### Примеры

c.in	c.out
0	1
1	4
2	12

## Разбор задачи С. Пути на доске

Пишем рекурсивную функцию вида “полный перебор”. Эта функция работает за  $O(Answer)$  и уже для  $n = 19, 20$  получает Time Limit. Добавляем предподсчет (заранее на своем компьютере считаем все ответы для  $n = 1..22$ ), получаем Accepted.

P.S. Предполагалось, что после лекции решение будет очевидно.

## Задача D. Тестирование шаффл-машин

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

У Сергея Михайловича очень интересная работа. Он занимается тестированием шаффл-машин — механизмов, используемых для тасования стопок карточек.

Машины, которые тестирует Сергей Михайлович, осуществляют перестановку стопок из  $N$  карточек, где  $N$  — чётное натуральное число. Механизм, по которому они работают, состоит в применении к исходной стопке определённой последовательности преобразований, каждое из которых имеет один из двух типов —  $U$  или  $D$ .

$U$ -преобразование производится следующим образом. Сначала исходная стопка из  $N$  карточек делится на две части, первая из которых состоит из  $N/2$  верхних карточек, а вторая — из  $N/2$  нижних. Затем в результирующую стопку поочерёдно помещается по одной карточке из каждой из двух частей, начиная с первой.  $D$ -преобразование отличается от  $U$ -преобразования только тем, что на втором шаге результирующая стопка начинает формироваться, начиная не с первой части, а со второй.

Если мы пронумеруем карточки сверху вниз числами  $1, 2, \dots, N$ , то в результате  $U$ -преобразования карточки будут следовать сверху вниз в порядке  $1, N/2+1, 2, N/2+2, \dots, N/2, N$ , а в результате  $D$ -преобразования порядок карточек будет таким:  $N/2 + 1, 1, N/2 + 2, 2, \dots, N, N/2$ .

Сергей Михайлович проводит тестирование следующим образом. Он берёт  $N$  карточек, пронумерованных от 1 до  $N$ , и формирует из них стопку так, чтобы номера карточек в стопке возрастили при их просмотре сверху вниз. Затем он помещает стопку в машину и производит её перетасовку. После этого Сергей Михайлович достает из результирующей стопки  $K$ -ю сверху карточку и в зависимости от её номера делает вывод об исправности или неисправности тестируемой машины.

Для ускорения процесса тестирования Сергею Михайловичу нужна программа, вычисляющая, чему будет равен номер  $K$ -й сверху карточки в результирующей стопке, если шаффл-машина работает корректно.

### Формат входного файла

Первая строка ввода содержит целые числа  $N$  и  $K$  ( $1 \leq K \leq N \leq 2\,000\,000\,000$ , число  $N$  — чётное). Во второй строке записано от 1 до 1000 символов ‘U’ и ‘D’ без пробелов. Эти символы описывают последова-

тельность преобразований, применяемых машиной для перестановки карточек. Символ ‘U’ соответствует  $U$ -преобразованию, а символ ‘D’ —  $D$ -преобразованию.

### Формат выходного файла

Единственная строка вывода должна содержать одно целое число — номер  $K$ -й сверху карточки в результирующей стопке, считая с единицы.

### Пример

d.in	d.out
20 7 DUUUDUDUDU	1

### Разбор задачи D. Тестирование шаффл-машин

Достаточно научиться за  $O(1)$  обращать обе операции перемешивания карт. Тогда получится решение исходной задачи за  $O(\text{длины строки})$ .

Обращения операции U:  $(i \bmod 2) * (\text{N div } 2) + (i \text{ div } 2)$

Обращения операции D:  $(1 - (i \bmod 2)) * (\text{N div } 2) + (i \text{ div } 2)$

### Задача Е. Белка и бамбук

Имя входного файла:	e.in
Имя выходного файла:	e.out
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

### Полное

Белка решила отправиться в кругосветное путешествие. Попав в тропики, она обнаружила, что жёлуди находить стало труднее. Зато она нашла отличный стебель бамбука, и теперь вместо того, чтобы каждый день таскать жёлуди по одному от одного дупла до другого, носит их с собой в бамбуке.

Бамбук — это трубка, один конец которой закрыт, а с другого конца можно класть или вынимать жёлуди. Диаметр трубки достаточно мал, поэтому если положить в неё жёлуди в определённом порядке, вынимать их можно только в обратном порядке.

Когда белка находит жёлудь, она сразу кладёт его в бамбук. Кроме того, время от времени голод заставляет белку достать один жёлудь из бамбука и съесть его; из-за устройства бамбука это будет тот из желудей в нём, который белка нашла позже всего.

Белка очень любит копить жёлуди в бамбуке. Поэтому каждый раз, когда приходится доставать из бамбука очередной жёлудь, она испытывает печаль. Однако мы знаем, как можно её утешить! Жёлуди характеризуются для белки *качеством* — целым числом от 1 до  $10^6$ . Когда белка достала очередной жёлудь, ей будет приятно знать, каково максимальное значение качества для всех желудей, которые в бамбуке ещё остались. Ваша задача состоит в том, чтобы снабдить её такой информацией.

### Формат входного файла

В первой строке ввода содержится одно целое число  $n$  — количество событий ( $1 \leq n \leq 100\,000$ ). Каждая из следующих  $n$  строк содержит по числу, описывающему событие. Если число положительное, то оно означает, что белка нашла жёлудь и положила его в свой бамбук. Если же число равно нулю, то оно означает, что белка проголодалась и вынула один жёлудь из бамбука. Числа во входном файле целые и не превосходят  $10^6$ . Гарантируется, что после первого же запроса бамбук никогда не пуст.

### Формат выходного файла

Для каждого доставания жёлудя из бамбука выведите строку, содержащую одно целое число — максимальное значение качества для всех желудей, оставшихся в этот момент в бамбуке.

### Пример

e.in	e.out
8	3
3	4
2	3
4	
0	
4	
3	
0	
0	

### Разбор задачи Е. Белка и бамбук

Нужно реализовать стэк с операцией минимум за  $O(1)$ .

Для этого кроме стэка элементов  $a$  будем хранить стэк минимумов на префиксах  $m$ .  $m[i + 1] = \min(m[i], a[i])$ .

## Задача F. Снеговики

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

Зима. 2012 год. На фоне грядущего Апокалипсиса и конца света незамеченной прошла новость об очередном прорыве в областях клонирования и снеговиков: клонирования снеговиков. Вы конечно знаете, но мы вам напомним, что снеговик состоит из нуля или более вертикально поставленных друг на друга шаров, а клонирование — это процесс создания идентичной копии (клона).

В местечке Местячково учитель Андрей Сергеевич Учитель купил через интернет-магазин “Интернет-магазин аппаратов клонирования” аппарат для клонирования снеговиков. Теперь дети могут играть и даже играют во дворе в следующую игру. Время от времени один из них выбирает понравившегося снеговика, клонирует его и:

- либо добавляет ему сверху один шар;
- либо удаляет из него верхний шар (если снеговик не пустой).

Учитель Андрей Сергеевич Учитель записал последовательность действий и теперь хочет узнать суммарную массу всех построенных снеговиков.

### Формат входного файла

Первая строка содержит количество действий  $n$  ( $1 \leq n \leq 200\,000$ ). В строке номер  $i + 1$  содержится описание действия  $i$ :

- $t \ m$  — клонировать снеговика номер  $t$  ( $0 \leq t < i$ ) и добавить сверху шар массой  $m$  ( $0 < m \leq 1000$ );
- $t \ \emptyset$  — клонировать снеговика номер  $t$  ( $0 \leq t < i$ ) и удалить верхний шар. Гарантируется, что снеговик  $t$  не пустой.

В результате действия  $i$ , описанного в строке  $i + 1$  создается снеговик номер  $i$ . Изначально имеется пустой снеговик с номером ноль.

Все числа во входном файле целые.

### Формат выходного файла

Выведите суммарную массу построенных снеговиков.

## Примеры

<b>f.in</b>	<b>f.out</b>
8	74
0 1	
1 5	
2 4	
3 2	
4 3	
5 0	
6 6	
1 0	

## Разбор задачи F. Снеговики

В данной задаче нужно реализовать так называемый Persistent Stack. По сути, нужно построить подвешенное за корень (изначальный пустой стек) дерево. Каждому стеку соответствует вершина.

Операция `rop` — перейти от текущей вершины к отцу. Операция `push` — создать новый лист, отцом листа сделать текущую вершину.

## Задача G. Вершинное покрытие

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

Давайте научимся решать следующую NP-задачу.

Вам дан граф и число  $K$ . Нужно найти в этом графе такое множество из  $K$  вершин, что для каждого ребра данного графа хотя бы один из двух концов ребра лежит в выбранном множестве вершин.

### Формат входного файла

На первой строке 3 числа —  $N, M, K$ . Где  $N$  — число вершин в графе,  $M$  — число ребер в графе.  $K$  — число  $K$  из условия.

Следующие  $M$  строк содержат описание ребер графа.

$$1 \leq K \leq N \leq 2000$$

$$1 \leq M \leq 10^5$$

Степень каждой вершины не меньше 1.

## Формат выходного файла

Если требуемое множество существует, на первой строке выведите слово Yes, а на второй  $K$  чисел — номера выбранных вами вершин.

Если же множества не существует, выведите слово No.

Если возможных ответов несколько, выведите любой.

Одну вершину нельзя брать в множество 2 раза. Т.е. все  $K$  чисел должны быть различны.

## Пример

g.in	g.out
2 1 2 1 2	Yes 1 2
3 3 1 1 2 2 3 3 1	No
7 6 2 1 2 1 3 1 4 2 5 2 6 2 7	Yes 1 2
5 5 2 1 2 1 3 1 4 1 5 4 5	Yes 4 1

## Разбор задачи G. Вершинное покрытие

Нужно найти множество из  $K$  вершин, покрывающее все ребра.

Если есть непокрытое ребро, то обязательно нужно взять в множество хотя бы один из концов этого ребра.

Таким образом мы уже получили решение за  $O(2^k * \text{Polynom}(n, m))$ .

Теперь заметим, что если какую-то вершину мы не берем, обязательно нужно брать всех ее соседей. Если вершина степени 1, всегда выгодно взять не ее, а ее соседа.

Т.е.  $Time(K) = Time(K - 1) + Time(K - 2)$  ( $K - 1$  — случай, когда мы берем вершину,  $K - deg$  — случай когда мы вершину не берем.  $deg \geq 2$ ).

## Задача Н. Праздничные дни

Имя входного файла:	<code>h.in</code>
Имя выходного файла:	<code>h.out</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Жители Флатландии очень любят праздники. Они считают, что любая более-менее важная памятная дата заслуживает того, чтобы ее отметить. Поэтому во Флатландии очень много государственных праздников. Настолько много, что там нет даже обычных выходных (как суббота и воскресенье в России) — жителям вполне хватает государственных праздников.

У каждого праздника есть день, когда он должен отмечаться. При этом день многих праздников определяется по очень сложным правилам; например, День защиты Флатландских волков отмечается в последнее полнолуние года, Международный День СУНЦов — в 42-й день летних каникул, а День Флатландского метеоролога — на следующий день после первого дождливого летнего четверга. (Правда, флатландские метеорологи настолько суровы, что могут предсказать, на какой день придется их профессиональный праздник, за год до него.)

Естественно, что при такой сложной и запутанной системе праздников нередко случается так, что несколько праздников выпадают на один и тот же день. Но флатландцы — очень систематичные люди, и они не могут отмечать несколько праздников в один и тот же день, поэтому в таких случаях праздники надо переносить.

Естественно, просто так переносить праздники нельзя. Перенос праздника допускается только при условии, что все дни между правильной датой праздника и тем днем, на который этот праздник переносится, также будут праздничными. Строго говоря, праздник с дня  $i$  можно отмечать в день  $j$  при одном из двух следующих условий:

- или  $i = j$ ,
- или  $i \neq j$ , но при этом в день  $i$ , а также во все дни между  $i$  и  $j$  невключительно отмечаются какие-то другие праздники (обратите внимание, что может быть как  $i < j$ , так и  $i > j$ ).

Таким образом, весь год будет состоять из чередующихся блоков рабочих и выходных дней, и перенос праздника допустим только при условии, что он попадает в тот же блок.

Флатландцы — последовательные люди, и придерживаются принципа «работать — так работать, отдохнуть — так отдыхать». Поэтому они не любят частых смен выходных и рабочих дней. Поэтому праздники должны быть перенесены так, чтобы средняя длина блоков выходных дней была наибольшей.

Сегодня во Флатландии, как и во всем мире, наступает Новый год. Через несколько часов президент Флатландии выступит с обращением перед гражданами. По традиции, он должен озвучить календарь выходных и рабочих дней на наступающий год. Это расписание должен был подготовить Министр Флатландии по Праздникам, но, к сожалению, вчера отмечался День Министров Флатландии, и потому Министр не успел подготовить календарь — и не успеет, поскольку компьютером пользоваться он не умеет, а вручную составление календаря отнимает много времени.

Поэтому подготовить календарь поручено вам. Напишите программу, которая, зная, на какой день сколько праздников приходится, перенесет их в соответствии с правилами. Учтите, что нельзя праздники отмечать раньше первого дня года, и нельзя их отмечать позже последнего дня.

## Формат входного файла

На первой строке входного файла находится одно число  $n$  — количество дней во Флатландском году ( $1 \leq n \leq 4000$ ). На второй строке находятся  $n$  неотрицательных целых чисел,  $i$ -ое из которых указывает, сколько праздников приходится на  $i$ -ый день года.

Гарантируется, что сумма всех чисел во второй строке входного файла не превосходит  $n$ .

## Формат выходного файла

Выведите в выходной файл  $n$  символов: для каждого дня года выведите, должен он быть выходным или рабочим. Выводите символ ‘X’ (латинская заглавная буква X) для рабочего дня и символ ‘\*’ (звездочка) для выходного.

## Примеры

<b>h.in</b>	<b>h.out</b>
6 0 0 2 0 1 0	XX***X
6 0 0 2 0 0 1	X**XX*

## Разбор задачи Н. Праздничные дни

Для каждого дня посчитаем число  $a[i]$  — число праздников в день  $i$ . По условию задачи нужно минимизировать число отрезков праздничных дней. Утверждается, что отрезок праздничных дней должен обладать только тем свойством, что сумма чисел  $a[i]$  на нем равна длине отрезка.

Динамика за  $O(n^2)$ . Состояние: мы уже распределили все праздники для дней  $[1..i]$ . Переход: или пропускаем один день, или добавляем отрезок праздничных дней. Проверка “хорошести” отрезка осуществляется за  $O(1)$  (мы знаем все частичные суммы).

## Задача I. Самый длинный путь

Имя входного файла:	<code>i.in</code>
Имя выходного файла:	<code>i.out</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 мб

В данном ориентированном графе найдите самый длинный путь такой, что каждая вершина графа встречается в нём не более одного раза.

### Формат входного файла

В первой строке входного файла заданы через пробел два целых числа  $n$  и  $m$  ( $1 \leq n \leq 22$ ,  $0 \leq m \leq 1000$ ). В следующих  $m$  строках заданы рёбра графа в формате  $u_i \ v_i$  — номера начальной и конечной вершин ребра  $i$ , соответственно. Граф может содержать петли и кратные рёбра.

### Формат выходного файла

В первой строке выходного файла выведите длину искомого пути  $l$ . Во второй строке выведите  $l+1$  число через пробел — вершины пути в порядке обхода. Если оптимальных ответов несколько, можно вывести любой из них.

## Примеры

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

## Разбор задачи I. Самый длинный путь

Одно из возможных решений этой задачи — динамика по подмножествам за  $O(2^n * n)$  с  $O(2^n)$  памяти. Состояние: множество посещенных вершин  $A$  и последняя вершина в пути  $v$ . Для каждого состояния мы храним один бит — достижимо ли это состояние. Итого памяти нам нужно  $O(2^n)$   $n$ -битовых чисел. Переход: для каждого из  $2^n$  множеств  $A$  у нас есть  $n$ -битовое число  $x[A]$  — вершины, на которые может оканчиваться путь. Пусть мы хотим перейти в вершину  $i$  из множества  $A$ .  $c[x[A]]$  — множество вершин связанных с хотя бы одной из  $x[A]$ , перейти в вершину  $i$  мы сможем тогда и только тогда, когда  $c[x[A]]$  содержит  $i$ . Это можно проверить за  $O(1)$ .

Подразумевалось тем не менее, что эта задача — упражнение на перебор с отсечениями. Для перебора нужно придумать состояние из динамики (отсечение меморизацией), но переход изобретать не нужно. Вместо красивого быстрого перехода мы на каждом шаге будем делать отсечение по ответу. Т.е. проверять, сколько вершин ( $x$ ) достижимо из конца пути и если  $CurrentLength + x \leq Answer$  сразу выходить из рекурсии. Полезно так же при переходе отсортировать вершины по убыванию  $x$ , но проходило решение и без этого отсечения.

## Задача J. Сумма не без разнообразия

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

Задана последовательность целых чисел  $A_1, A_2, \dots, A_N$ .

Необходимо выбрать из нее подпоследовательность из подряд стоящих чисел  $A_i, A_{i+1}, \dots, A_j$  так, чтобы она содержала не менее  $K$  различных чисел, и сумма  $S = A_i + A_{i+1} + \dots + A_j$  была максимальной.

### Формат входного файла

Первая строка ввода содержит целые числа  $N$  и  $K$  ( $1 \leq K \leq N \leq 200\,000$ ).

Вторая строка содержит  $N$  целых чисел  $A_1, A_2, \dots, A_N$  ( $|A_i| \leq 1\,000\,000\,000$ ).

### Формат выходного файла

В первой строке необходимо вывести максимальное возможное значение суммы  $S$ . Во второй строке выведите индексы первого и последнего элементов найденной оптимальной подпоследовательности. Если существует несколько решений, подойдет любое из них.

Если не существует подпоследовательностей, удовлетворяющих решению задачи, выведите одну строку со словом “IMPOSSIBLE” (без кавычек).

### Примеры

<b>j.in</b>	<b>j.out</b>
7 3	-89
-99 1 2 -100 3 2 3	2 7
3 2	IMPOSSIBLE
1 1 1	

## Разбор задачи J. Сумма не без разнообразия

Решение за  $O(n)$  методом двух указателей. Перебираем правую границу отрезка  $R$  (первый указатель). Выбираем максимальную левую границу  $maxL$  (второй указатель) такую, что на отрезке есть  $K$  различных чисел. Далее левую границу  $L$  нужно выбрать на отрезке  $[1..maxL]$  такую, что  $Sum[1..L - 1] = Min$ , т.к.  $Sum[L..R] = Sum[1..R] - Sum[1..L - 1]$ .

Это можно сделать за  $O(1)$ , предподсчитав суммы и минимумы сумм на префиксах.

## Задача К. Валидатор к игре в тетрис

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

Напомним условие задачи **Тетрис**:

*В процессе игры в Тетрис связные фигуры (т. е. связные, если идти по 4-м направлениям) падают вниз в колодец из  $N$  строк и 20 столбцов. Каждая фигура помечена уникальной буквой английского алфавита от “A” до “Z”.*

*Фигуры, пока они падают, ни двигать, ни вращать нельзя.*

*Ваша программа должна по описанию колодца определить, в каком порядке падали блоки.*

### Формат входных данных

*Первая строка входного файла содержит целое число  $N$  ( $0 \leq N \leq 50$ ) – количество строк в колодце. Каждая из следующих строк содержит по 20 символов, каждый из которых – это либо буква от “A” до “Z”, либо символ “.” (ASCII 46), обозначающий пустую клетку.*

Вам требуется написать **валидатор тестов** к этой задаче

### Формат входного файла

Входной файл содержит несколько тестов. Каждый тест дан в следующем формате:

Число  $N$  ( $1 \leq N \leq 50$ ) и  $N$  строк из 20 символов каждая.

Символы могут иметь ASCII коды от 32 до 127.

Сумма  $N$  по всем тестам не превысит 500 000.

### Формат выходного файла

Для каждого теста выведите YES или NO – корректен тест или нет.

Тест считается корректным, если выполнены все следующие условия:

- Используются только символы “A..Z” и “.”

- Фигура (т.е. множество всех клеток с некоторой буквой) образует связную область.
- Никакая фигура не “*висит в воздухе*”.
- Фигуры действительно могли упасть в таком порядке.

## Пример

<b>k.in</b>	<b>k.out</b>
6	YES
.....XX.....	NO
.....MMM.....	NO
.....K.....	NO
.....KKK.....	NO
....ZAAA.FFF.....	YES
....ZZZA..F.B.....	
2	
??.	
??.	
3	
AAB.....	
ABB.....	
AAB.....	
3	
.....	
AA.....	
..B.....	
1	
A.....A	
1	
ABCDEFGHIJKLMNPQRST	

## Разбор задачи К. Валидатор к игре в тетрис

Можно рассмотреть орграф, вершины которого — фигуры. Ребро между фигурами есть, если одна из них должна падать точно до другой. Тогда задача заключается в том, чтобы за  $O(WH)$  построить этот граф и за  $O(E)$  проверить его на ацикличность.

Кроме этого, конечно, нужно проверить, что все символы корректны, все фигуры связны и никакая фигура не висит в воздухе.

Для каждого двух соседних клеток, в которых есть фигуры разного типа, проведем ребро. Утверждается, что граф, который мы ищем — транзитивное замыкание полученного графа. Наличие в графе циклов после транзитивного замыкания не меняется, поэтому достаточно на ацикличность проверить уже полученный граф.

Асимптотика решения на одном тесте —  $O(WH)$ . Time Limit можно было получить или обнуляя большой массив на каждом тесте, или решая один тест дольше чем  $O(E)$ . Например,  $O(N^3)$  (даже несмотря на то, что  $N \leq 26$ ).

## **Задача L. Возьми себе правило — летай всегда GraphAero!**

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

Наконец, авиаперевозки стали доступны всем и каждому! Однако, из-за жёсткой конкуренции в сфере пассажироперевозок осталось только две авиакомпании: “GraphAero Airlines” и “Aerofloat”.

Авиакомпания “GraphAero Airlines” активно развивается. Ведь для получения большей прибыли... простите, для удобства пассажиров каждый месяц компания добавляет один новый рейс. Компании “Aerofloat” остаётся довольствоваться тем, что остаётся. А именно, единственная возможность удержаться на плаву — добавлять рейсы, дублирующие самые загруженные рейсы компании “GraphAero Airlines”. Рейс является самым загруженным, если существует такая пара городов, что можно долететь (возможно, с пересадками) из одного города в другой, используя рейсы авиакомпании, но если этот рейс отменить — то долететь будет невозможно. Аналитикам компании “Aerofloat” необходимо постоянно контролировать ситуацию — сколько в данный момент существует самых загруженных рейсов.

Поскольку вы уже давно мечтаете летать по льготным ценам (скидка 10<sup>-5</sup>%), вы решили оказать посильную помощь. Помните: самолёты летают по всему миру! Между двумя крупными городами может быть более одного рейса, а города бывают настолько большими, что самолёты могут летать в пределах одного города. Рейсами можно пользоваться как в одну, так и в другую сторону.

### **Формат входного файла**

Первая строка входного файла содержит целое число  $N$

( $1 \leq N \leq 100\,000$ ) — количество городов и  $M$  ( $0 \leq M \leq 100\,000$ ) — изначальное число рейсов компании “GraphAero Airlines”. Далее следует  $M$  строк, в каждой содержится описание очередного рейса — номера двух городов, между которыми осуществляется рейс. В следующей строке содержится число  $K$  ( $1 \leq K \leq 100\,000$ ) — количество добавленных рейсов. Далее содержится описание добавленных рейсов в таком же формате.

### Формат выходного файла

После каждого добавления нового рейса выведите на отдельной строке одно число — количество самых загруженных рейсов.

1.in	1.out
4 0	1
4	2
1 2	3
2 3	0
3 4	
1 4	
4 3	3
1 2	2
2 3	1
3 4	0
4	
1 1	
1 2	
1 3	
1 4	

### Разбор задачи L. Возьми себе за правило — летай всегда GraphAero!

Перефразируем задачу: изначально граф пустой, в него добавляются ребра. После каждого запроса нужно говорить, сколько в графе мостов.

Пусть граф уже связан. Тогда будем поддерживать дерево компонент реберной двусвязности. Дерево будет подвешенным за корень. Когда мы добавляем в граф ребро, в этом дереве образуется цикл. Выделим его за  $O(\text{длины цикла})$  и сожмем в одну вершину с помощью СНМ.

Пускай граф не связан, тогда мы храним не одно дерево, а целый лес компонент двусвязности. Добавляется один новый запрос — объединить два разных дерева. Перед запросом есть два дерева, у них есть два корня, нужно оставить один корень, т.е. в одном из деревьев нужно поменять

ориентацию ребер. Выберем меньшее из деревьев и поменяем в нем в лоб, за  $O(\text{размера дерева})$ .

Суммарное время на такие операции =  $O(N \log N)$  просмотров вершин т.к. для каждой просмотренной вершины верно, что размер дерева, в котором она теперь живет, удвоился.

## Задача М. Тест к задаче про Клики

Имя входного файла: **m.in**  
Имя выходного файла: **m.out**  
Ограничение по времени: 1 с  
Ограничение по памяти: 64 Мб

В этой задаче Вам предлагается решение для задачи *Клики*.

Данное решение, естественно, не получает *Accepted*, но работает все же за время  $o(2^N)$ .

```
// i - текущая вершина, p - текущее множество вершин
long long go( int i, long long p )
{
    if (p == 0) // если множество p пусто
        return 1;

    counter++; // время работы программы
    while (((p >> i) & 1) == 0) // лежит ли вершина i в множестве p
        i++;
    p ^= 1LL << i; // исключить вершину i из множества p
    if ((p & c[i]) == p) // c[i] - множество вершин, соединенных
        // с вершиной i ребром
        return 2 * go(i, p);
    // p & c[i] - пересечение множеств p и c[i]
    return go(i, p) + go(i, p & c[i]);
}

counter = 0; // обнуляем время работы программы
result = go(0, (1LL << n) - 1); // запускаемся от множества всех вершин
```

Ваша задача — построить тест, на котором это решение не уложится в *Time Limit*. То, сколько работает программа, определяется значением переменной *counter*.

## Формат входного файла

Число  $N$  — количество вершин в графе, который Вам требуется построить.

Всего в задаче 2 теста.

Первый тест содержит  $N = 20$ . На нем переменная *counter* к концу работы программы должна быть не меньше 5 000

Второй тест содержит  $N = 40$ . На нем переменная *counter* к концу работы программы должна быть не меньше 50 000 000

### Формат выходного файла

Выведите тест. Тест должен соответствовать формату входных данных задачи *Клики*.

### Пример

<b>m.in</b>	<b>m.out</b>
20	20 01100010101101010001 10010010111010001111 1000001001111000001 0100000000000100100 0000001100111001011 00000000011011011101 11101001101011010101 0000101001000000101 11001010000101110110 01100101001001010010 11100110010000010011 10101000100001100001 01101110000000101110 10101110110100100000 00010000100111010101 10000110111000101100 01001100000010010110 01010111100010111011 01001000111010001100 11101111001100100100

### Примечание

Ответ из примера получит вердикт *Wrong Answer*, т.к. переменная *counter* к концу работы программы будет иметь значение 213.

### Разбор задачи М. Тест к задаче про Клики

Ответ = полный граф - ребра  $(2i)$  to  $(2i + 1)$ .

Время работы алгоритма на этом тесте  $\geq 3^{\frac{n}{2}}$

## Задача N. Задача про Клики

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

Дан неориентированный, невзвешенный граф. Нужно найти количество подклика данного графа. Подклика — подграф, являющийся полным графом. У полного графа из  $V$  вершин подклак ровно  $2^V$ . У пустого графа из  $V$  вершин подклак ровно  $V + 1$ .

### Формат входного файла

Число  $V$  ( $1 \leq V \leq 60$ ) — количество вершин в графе.

Далее в  $V$  строках графа задана матрица смежности графа. 0 обозначает отсутствие ребра, соответственно 1 обозначает присутствие ребра. На главной диагонали всегда стоят нули. Матрица симметрична.

### Формат выходного файла

Число подклика данного графа.

### Пример

n.in	n.out
6	
011100	
101100	
110100	
111000	
000001	
000010	
	19

## Разбор задачи N. Задача про Клики

Замечание: поскольку  $n \leq 60$ , любое множество задается целым 64-битным числом, и любая операция с множествами выполняется за  $O(1)$ .

### Решение задачи — перебор.

Перебираем вершины в каком-то заранее фиксированном порядке, очередную вершину или берем в клику, или не берем. В каждый момент времени поддерживаем множество  $C$  — те вершины, которые еще можно добавить в клику (т.е. те вершины, которые связаны со всеми уже добавленными).

Если  $C$  пустое, return 1

Если  $C$  состоит ровно из одного элемента, return 2

Добавим отсечение меморизацией, будем хранить  $f[C]$ .

**Докажем, что даже такое решение работает за  $O(2^{n/2})$ .**

Рассмотрим первые  $\frac{n}{2}$  вершин. Перебирая их, мы разберем не более  $2^{\frac{n}{2}}$  вариантов. При этом после просмотра первых  $\frac{n}{2}$  вершин множество  $C$  их уже точно не содержит. Т.е. различных множеств  $C$  от этого момента до конца алгоритма не более  $2^{\frac{n}{2}}$ . Меморизация позволяет для каждого множества  $C$  посчитать  $f[C]$  только один раз. Время работы оказывается не  $2^{\frac{n}{2}} * 2^{\frac{n}{2}}$ , а  $2^{\frac{n}{2}} + 2^{\frac{n}{2}}$ . Получилось что-то похожее на Meet-In-The-Middle, но эту идею мы используем только для доказательства времени работы, наше решение — перебор.

Сейчас мы перебирали вершины в произвольном порядке. Теперь выберем более оптимальный порядок вершин.

**Более оптимальный порядок вершин:**

Изначально множество вершин, которые мы “взяли” пусто. Будем его наращивать. Вершины добавятся в множество в каком-то порядке. В этом же порядке и будем перебирать их в дальнейшей рекурсии. Алгоритм, в каком порядке брать вершины:

1. Если есть вершина, связанная со всеми оставшимися, берем ее.
2. Если есть 2 вершины, связанные со всеми, берем их.
3. Если есть 3 вершины, связанные со всеми, берем их :-).
4. Иначе берем 4 вершины, между которыми как можно меньше ребер.

**Докажем, что получившееся решение работает за  $O(2^{3n/7})$ .**

Первые три пункта (благодаря меморизации) не ветвят рекурсию (множество  $C$  поменяется одинаково).

Четвертый пункт берет 4 вершины. Есть в худшем случае  $2^4 = 16$  вариантов “брать или не брать” эти вершины. Но ребер между этими 4-мя вершинами будет не более 3-х ребер (благодаря первым 3-м пунктам). Из этого следует, что вариантов на самом деле не 16, а всего 8.

Теперь оценка времени работы: пусть мы выбрали  $k$  четверок, тогда время работы не более чем  $2^{3k} + 2^{n-4k}$ . Выберем  $k$  такое, что  $3k = n - 4k$  и получим требуемое время  $O(2^{3n/7})$ .

**Эвристики.**

Выше представлены асимптотические строгие решения. Кроме них работали различные (но далеко не все) эвристики, в разных местах выбирающие вершины *min* или *max* степени. Некоторые эвристики перебирали вершины НЕ в фиксированном, а в меняющемся по ходу рекурсии порядке.

**Конец.**

## Задача О. Connect and Disconnect

Имя входного файла:	o.in
Имя выходного файла:	o.out
Ограничение по времени:	3 с (4 с для Java)
Ограничение по памяти:	256 Мб

Знаете ли Вы что-нибудь про DFS, Depth First Search, поиск в глубину?

Например, используя этот метод, можно проверить связность графа за время  $O(E)$ . Можно даже за тоже самое время посчитать число компонент связности графа.

Знаете ли Вы что-нибудь про DSU, Disjoint Set Union, систему непересекающихся множеств? Используя эту структуру данных, Вы можете обрабатывать запросы вида “Добавить ребро в граф” и “Посчитать число компонент связности графа” быстро. Т.е. оба запроса за  $O(\log N)$ .

А знаете ли Вы что-нибудь про Dynamic Connectivity Problem? В этой задаче, вам нужно уметь быстро обрабатывать 3 типа запросов:

1. Добавить ребро в граф.
2. Удалить ребро из графа.
3. Посчитать число компонент связности графа.

### Формат входного файла

Изначально граф пуст.

Первая строка входного файла содержит 2 числа —  $N$  и  $K$  — число вершин и число запросов ( $1 \leq N \leq 300\,000$ ,  $0 \leq K \leq 300\,000$ ).

Следующие  $K$  содержат запросы, по одному в строке. Есть 3 типа запросов:

1.  $+ u v$ : добавить ребро между вершинами  $u$  и  $v$ . Гарантируется, что в момент получения запроса такого ребра в графе нет.
2.  $- u v$ : удалить ребро между вершинами  $u$  и  $v$ . Гарантируется, что в момент получения запроса такое ребро в графе есть.
3.  $?$ : посчитать число компонент связности в графе в текущий момент.

Вершины нумеруются от 1 до  $N$ . Нет запросов с  $u = v$ . Граф неориентированный.

### Формат выходного файла

Для каждого запроса вида ‘?’ выведите число компонент связности графа в момент времени запроса на отдельной строке.

## Пример

<b>o.in</b>	<b>o.out</b>
5 11	5
?	1
+ 1 2	1
+ 2 3	2
+ 3 4	
+ 4 5	
+ 5 1	
?	
- 2 3	
?	
- 4 5	
?	

## Разбор задачи O. Connect and Disconnect

Решение за  $O(K \log K \log N)$ .

Основная идея — не будем выполнять запросы Del. Будем НЕ выполнять парные к ним Add.

Строим дерево отрезков на запросах. Путешествуем по дереву отрезков рекурсивно сверху вниз. Пусть состояние =  $[L..R]$ . Обрабатывая это состояние, мы узнаем, сколько мостов было после запросов  $[L..R]$ . В состоянии  $[L..R]$  все запросы Add с индексом больше  $R$  мы уже точно не будем выполнять, а про все запросы Add, парный к которым Del находится НЕ в  $[L..R]$ , мы уже точно знаем, нужно ли их отменять (т.е. правда ли, что 100% эти запросы Add нужно делать или, что 100% их делать не нужно).

Все Add-ы, про которые мы в состоянии  $[L..R]$  ничего сказать пока не можем, храним. При переходе к более мелкому отрезку это множество будет уменьшаться, пересчитывать мы его будем в лоб за линейное время.

P.S. Подразумевалось, что на лекции перед туром давалось общее описание решения задачи.

## Задача Р. Кривые зеркала

Имя входного файла:	p.in
Имя выходного файла:	p.out
Ограничение по времени:	0.5 с
Ограничение по памяти:	64 Мб

За один залп BFG-9000 уничтожает монстров на трех соседних балконах

чиках. Балкончики при этом не разрушаются. А монстры погибают. ( $N$ -й балкончик соседствует с первым). После залпа оставшиеся в живых монстры наносят Леониду (главному герою романа) повреждения — по одной единице каждый. Далее следует новый залп и так до тех пор, пока все монстры не погибнут. Требуется определить минимальные повреждения, которые может понести Леонид.

### Формат входного файла

Первая строка содержит целое число  $N$ , количество балкончиков, на которых монстры заняли круговую оборону ( $3 \leq N \leq 30$ ). Во второй строке даны  $N$  целых чисел — количество монстров на каждом балкончике (на каждом не менее 1 и не более 100).

### Формат выходного файла

Выведите минимальное количество единиц повреждений.

### Пример

p.in	p.out
7	9
3 4 2 2 1 4 1	

## Разбор задачи Р. Кривые зеркала

Решение — перебор.

Предполагавшиеся оптимизации:

- Состояние — профиль  $2^n$ , какие из монстров все еще живы. Переход, разрушить балконы  $i, i + 1, i + 2$ . Каждый переход можно сделать за  $O(1)$ , суммарный урон, который наносят монстры то же пересчитывается за  $O(1)$ . Рассмотрим все  $n$  переходов, если какой-то мажорирует другой (уничтожает надмножество балконов), нужно из этих двух делать только один, более сильный.
- Меморизация с помощью хэш-таблицы. После этого решение работает за  $O(2^n \cdot n)$ .
- Отсечение по ответу  $CurrentDamage + F(State) \geq Answer \Rightarrow$  перебор из данного состояния не даст новых результатов.  $F(State)$  — функция оценки, какое минимальное повреждение нам точно нанесут.  $F(State)$  можно посчитать так: отсортировать всех живых монстров по убыванию силы и оптимистично считать, что за каждый ход мы будем убивать трех самых сильных монстров. За  $O(n \log n)$  посчитаем  $F(State)$ .

4. При переборе переходов давайте перебирать только  $M$  лучших, а по константе  $M$  делать *IterativeDeepening*. Это отсечение
5. Отсечение по времени тоже лишним не будет

### Задача Q. Сортировка вручную

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

Пришло время расставить книжки на полке. На ней стоят  $n$  книг, однозначно пронумерованных целыми числами от 1 до  $n$ . Требуется расставить их в порядке возрастания номеров. Вы прекрасно знаете, что *быстрая сортировка и сортировка вставками* — быстрые методы сортировки, однако их непросто исполнять вручную. Они эффективны для компьютеров, а не для людей.

Поэтому вы решили сортировать книги, вставляя  $i$ -ю книгу на  $i$ -ю позицию. Сколько операций вам придётся выполнить для успешной сортировки? Вот два примера операции:

- $1\ 3\ 4\ 5\ 2 \rightarrow 1\ 2\ 3\ 4\ 5$ , если мы вставляем 2 на свое место
- $1\ 3\ 4\ 5\ 2 \rightarrow 1\ 4\ 3\ 5\ 2$ , если мы вставляем 3 на свое место

### Формат входного файла

В первой строке входного файла содержится целое число  $n$  ( $1 \leq n \leq 20$ ) — количество книг на полке.

Во второй строке записано  $n$  различных целых чисел от 1 до  $n$  — номера книг в том порядке, в котором они расставлены вначале.

### Формат выходного файла

Выведите единственное целое число — минимальное количество операций указанного вида, которое придётся выполнить для сортировки.

## Примеры

<b>q.in</b>	<b>q.out</b>
3 1 2 3	0
3 2 1 3	1
3 3 2 1	2
9 7 2 9 4 6 8 5 3 1	5

## Разбор задачи Q. Сортировка вручную

Решение — перебор.

Предполагавшиеся оптимизации:

1. Состояние — текущий массив. От него у нас в каждый момент времени есть полиномиальный хэш.
2. Меморизация с помощью хэш-таблицы. От состояния хранится “минимальное время дойти до этого состояния”. Если это время не улучшилось с тех пор, как мы приходили в состояние в прошлый раз, рекурсия обрывается.
3. Отсечение по ответу  $CurrentTime + F(State) \geq Answer \Rightarrow$  перебор из данного состояния не даст новых результатов.  $F(State)$  — функция оценки, какое минимальное число операций еще точно предстоит сделать.  $F(State)$  можно посчитать так:  $n - LIS$ , где  $LIS$  — длина наибольшей возрастающей подпоследовательности.
4. dfs меняем на bfs с ограничением длины очереди (если очередь переполняется, берем только  $MAX\_SIZE$  лучших по  $F(State)$  состояний).
5. Отсечение по времени тоже лишним не будет.

## Задача R. СНМ

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

Ваша задача — реализовать **Persistent Disjoint-Set-Union**. Что это значит?

### Про Disjoint-Set-Union:

Изначально у вас есть  $n$  элементов. Нужно научиться отвечать на 2 типа запросов.

- +  $a$   $b$  — объединить множества, в которых лежат вершины  $a$  и  $b$ ;
- ?  $a$   $b$  — сказать, лежат ли вершины  $a$  и  $b$  сейчас в одном множестве.

### Про Persistent:

Теперь у нас будет несколько копий (версий) структуры данных **Disjoint-Set-Union**.

Запросы будут выглядеть так:

- +  $i$   $a$   $b$  — запрос к  $i$ -й структуре, объединить множества, в которых лежат вершины  $a$  и  $b$ . При этом  $i$ -я структура остается неизменной, создается новая версия, ей присваивается новый номер (какой? читайте дальше);
- ?  $i$   $a$   $b$  — запрос к  $i$ -й структуре, сказать, лежат ли вершины  $a$  и  $b$  сейчас в одном множестве.

## Формат входного файла

На первой строке 2 числа  $N$  ( $1 \leq N \leq 10^5$ ) и  $K$  ( $0 \leq K \leq 10^5$ ) — число элементов и число запросов. Изначально все элементы находятся в различных множествах. Эта изначальная копия (версия) структуры имеет номер 0.

Далее следуют  $K$  строк, на каждой описание очередного запроса. Формат запросов описан выше. Запросы нумеруются целыми числами от 1 до  $K$ .

При обработке  $j$ -го запроса вида +  $i$   $a$   $b$ , новая версия получит номер  $j$ . Запросов к несуществующим версиям не будет ( $j > i$ ).

## Формат выходного файла

Для каждого запроса вида ?  $i$   $a$   $b$  на отдельной строке нужно вывести YES или NO.

## Пример

r.in	r.out
4 7	NO
+ 0 1 2	YES
? 0 1 2	YES
? 1 1 2	YES
+ 1 2 3	NO
? 4 3 1	
? 0 4 4	
? 4 1 4	

## Разбор задачи R. СНМ

Решение — Persistent Array.

Хорошо известная реализация Системы Непересекающихся Множеств (СНМ) состоит из двух массивов — ранги вершин и родители вершин. Если оба массива сделать persistent, то и вся структура будет persistent.

В СНМ мы применяем 2 эвристики — сжатие путей и подвешивание меньшего дерева к большему. Сжатие путей ничего не дает в случае persistent структуры, так как нас интересует, сколько один запрос работает в худшем случае. И с, и без этой эвристики время на один запрос в худшем случае будет  $O(\log N)$ .

Собственно время работы всего решения  $O(K \log^2 N)$ .

## День шестой (17.02.2011 г.) Конкурс Виталия Неспирного и Антона Лунёва

### Об авторах...

**Неспирный Виталий Николаевич,** кандидат физико-математических наук (специальность 01.02.01 “Теоретическая механика”), докторант Института прикладной математики и механики НАН Украины, старший научный сотрудник отдела технической механики, ученый секретарь специализированного ученого совета по защите диссертаций на соискание ученого звания доктора наук (Д 11.193.01) по специальности 01.02.01, доцент кафедры теории упругости и вычислительной математики Донецкого национального университета, заместитель председателя жюри II-III этапов Всеукраинской олимпиады школьников по информатике в Донецкой области, 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:

- занял 1-е место на Всеукраинской студенческой олимпиады по информатике (Николаев, 2000);
- в составе команды математического факультета ДонНУ занял 1-е место на Всеукраинской ACM-олимпиаде по программированию и 8-е в личном первенстве (Винница, 2001);
- подготовил призеров международных школьных олимпиад – Петр Луференко (2002), Роман Ризванов (2006), Вадим Янушкевич (2009);
- совместно с А. И. Парамоновым подготовил команду DonNU United, занявшую 7-е (2008), 4-е (2009) и 7-е место (2010) в ACM SEERC.

**Лунёв Антон Андреевич**, аспирант Донецкого национального университета, специальность 01.01.01 “Математический анализ”.

Основные научные интересы:

- полнота и базисность систем корневых векторов некоторых классов обыкновенных дифференциальных операторов;
- точные константы в неравенствах для промежуточных производных;
- корневые решения обобщенного уравнения Маркова;
- игра Грюнди и ее обобщения;
- увлекается программированием.



Основные достижения:

- трижды (2005, 2006, 2007) был награжден первым призом на международной математической олимпиаде для студентов IMC;
- в 2006, 2007 годах награжден серебряной медалью, а в 2008 — золотой, на международной студенческой математической олимпиаде в Иране;
- участник команды по программированию Донецкого национального университета;
- в составе команды занял 7-е место в 2008 году, 4-е место в 2009 году и 7-е место в 2010 году на региональной олимпиаде ACM по программированию (Southeastern European Regional Contest). По итогам олимпиады 2010 года команда Донецкого национального университета будет представлять Юго-Восточный Европейский регион на Финале Мира по программированию;
- на Всеукраинской студенческой олимпиаде по программированию (г. Винница) команда заняла 4-е место в 2009 году и 2-е место в 2010 году;

- по итогам 8-го Открытого Кубка (осень 2010) команда награждена дипломом первой степени;
- награжден дипломом первой степени по итогам трёх последних серий индивидуальных соревнований по программированию SnarkNews Summer/Winter Series (Лето 2009 – 3-е место, Зима 2010 – 3-е место, Лето 2010 – 4-е место);
- участник онсайт-раунда Яндекс Open 2010.

## Теоретический материал. Близость точек на плоскости. Диаграмма Вороного

**1. Реберный список с двойными связями** Реберный список с двойными связями (РСДС) особенно удобен для представления планарного графа, уложенного на плоскости. Плоская укладка планарного графа  $G = (V, E)$  – это отображение каждой вершины из  $V$  в точку на плоскости, а каждого ребра из  $E$  – в простую линию, соединяющую пару образов концевых вершин этого ребра так, чтобы образы ребер пересекались только в своих концевых точках. Хорошо известно, что любой планарный граф можно уложить на плоскости так, чтобы все ребра отобразились в прямолинейные отрезки.

Пусть  $V = \{v_1, \dots, v_N\}$ , а  $E = \{e_1, \dots, e_M\}$ . Главная компонента РСДС для планарного графа  $(V, E)$  – это реберный узел. Между ребрами и реберными узлами существует взаимно однозначное соответствие, то есть каждое ребро представлено ровно один раз. Реберный узел содержит три информационных поля  $(V_1, V_2, F)$  и три поля указателей  $R$ ,  $P_1$  и  $P_2$ ; следовательно, соответствующую структуру данных легко реализовать как шесть массивов, каждый из которых состоит из  $M$  ячеек. Значения этих полей таковы. Поле  $V_1$  содержит начало ребра, а поле  $V_2$  содержит его конец; так ребро получает условленную ориентацию. Поле  $F$  будет содержать номер грани, расположенной справа от ребра. Указатель  $R$  будет указывать на обратное ребро – ведущее из  $V_2$  в  $V_1$ . Указатель  $P_1$  будет задавать реберный узел, ведущий в вершину  $V_1$  из некоторой вершины  $V_i$ , такой, что угол  $V_i V_1 V_2$  будет наименьшим при определении его величины против часовой стрелки. Аналогично,  $P_2$  задает реберный узел, ведущий из вершины  $V_2$  в некоторую вершину  $V_i$ , такую, что угол  $V_1 V_2 V_i$  наименьший. Имена граней и вершин могут быть заданы целыми числами.

С помощью РСДС легко можно вычислить ребра, инцидентные заданной вершине, или ребра, ограничивающие заданную грань. Для этого графа,

содержащего  $N$  вершин и  $F$  граней, можно ввести два массива  $HV[1 : N]$  и  $HF[1 : F]$  для заголовков списков этих вершин и граней. Эти массивы можно заполнить при просмотре массивов  $V1$  и  $F$  за время  $O(N)$ . При нахождении ребер, исходящих из заданной вершины, естественным будет их перечисление в порядке против часовой стрелки, а обход грани совершать по часовой стрелке. Однако за счет хранения ссылок на обратные ребра, нетрудно получить и обратный порядок.

**2. Объединение двух выпуклых оболочек.** Пусть заданы два выпуклых многоугольника  $P_1$  и  $P_2$ . Требуется найти их выпуклую оболочку. Для решения этой задачи воспользуемся следующим алгоритмом.

1. Найти некоторую внутреннюю точку  $p$  многоугольника  $P_1$ . (Например, его центр масс. Такая точка  $p$  будет внутренней точкой  $CH(P_1 \cup P_2)$ ).
2. Определить, является ли  $p$  внутренней точкой  $P_2$ . Это может быть сделано за время  $O(N)$ .
3. Если  $p$  является внутренней точкой  $P_2$ , то вершины (как  $P_1$ , так и  $P_2$ ) оказываются упорядоченными в соответствии со значением полярного угла относительно точки  $p$ . За время  $O(m + n)$  можно получить упорядоченный список вершин (как  $P_1$ , так и  $P_2$ ) путем слияния списков вершин этих многоугольников.
4. Если же  $p$  не является внутренней точкой  $P_2$ , тогда если смотреть из точки  $p$ , то многоугольник  $P_2$  лежит в клине с углом разворота меньшим или равным  $\pi$ . Этот клин определяется двумя вершинами  $u$  и  $v$  многоугольника  $P_2$ , которые могут быть найдены за линейное время за один обход вершин многоугольника  $P_2$ . Эти вершины разбивают границу  $P_2$  на две цепи вершин, являющихся монотонными относительно изменения полярного угла относительно  $p$ . При движении по одной цепи угол увеличивается, при движении по другой – уменьшается. Одна из этих двух цепей, являющаяся выпуклой по направлению к точке  $p$ , может быть немедленно удалена, так как ее вершины будут внутренними точками  $CH(P_1 \cup P_2)$ . Другая цепь  $P_2$  и граница  $P_1$  представляют два упорядоченных списка, содержащих в сумме не более  $m + n$  вершин. За время  $O(m + n)$  их можно слить в один список вершин, упорядоченных по углу относительно точки  $p$ .
5. Теперь к полученному списку можно применить метод обхода Грэхема, требующий лишь линейное время, в результате чего и получится выпуклая оболочка двух многоугольников.

Побочным результатом описанного метода слияния является вычисление (нахождение) “опорных прямых”, если они имеются, для двух выпуклых

многоугольников. Опорной прямой к выпуклому многоугольнику  $P$  называется прямая  $l$ , проходящая через некоторую вершину многоугольника  $P$  таким образом, что внутренность  $P$  целиком находится по одну сторону от  $l$ . Очевидно, что два выпуклых многоугольника  $P_1$  и  $P_2$  с  $m$  и  $n$  вершинами соответственно, таких, что один не лежит целиком внутри другого, имеют общие опорные прямые (по крайней мере не более  $2 \min(m, n)$ ). Если уже получена выпуклая оболочка объединения  $P_1$  и  $P_2$ , то опорные прямые вычисляются в результате просмотра списка вершин  $CH(P_1 \cup P_2)$ . Каждая пара последовательных вершин  $CH(P_1 \cup P_2)$ , одна из которых принадлежит  $P_1$ , а другая  $P_2$ , определяет опорную прямую.

**3. Области близости. Диаграмма Вороного.** Пусть на плоскости задано множество  $S$ , содержащее  $N$  точек. Требуется для каждой точки  $p_i$  множества  $S$  определить область точек  $(x, y)$  на плоскости, для которых расстояние до  $p_i$  меньше, чем до любой другой точки множества  $S$ .

Отметим, что интуитивно решение этой задачи представляется как построение разбиения плоскости на области, каждая из которых является множеством точек  $(x, y)$ , более близких к некоторой точке множества  $S$ , чем к любой другой точке  $S$ . Заметим также, что если такое разбиение плоскости известно, то, применив процедуру поиска, определяющую какой из областей разбиения принадлежит некоторая точка  $q$ , можно непосредственно получить решение задачи о поиске ближайшей точки из множества к заданной. Исследуем теперь структуру этого разбиения плоскости. Если имеются две точки  $p_i$  и  $p_j$ , то множество точек, более близких к  $p_i$ , чем к  $p_j$  есть не что иное, как полуплоскость, определяемая прямой, перпендикулярной отрезку  $p_i p_j$  и делящей его пополам, и содержащая точку  $p_i$ . Обозначим эту полуплоскость  $H(p_i, p_j)$ . Множество точек, более близких к  $p_i$ , чем к любой другой точке, которое будем обозначать  $V(i)$ , получается в результате пересечения  $N - 1$  полуплоскостей. Это множество является выпуклой многоугольной областью, имеющей не более  $N - 1$  сторон. Таким образом,

$$V(i) = \bigcap_{i \neq j} H(p_i, p_j).$$

Область  $V(i)$  называется многоугольником Вороного, соответствующим точке  $p_i$ .

Получаемые таким образом  $N$  областей образуют разбиение плоскости, представляющее некоторую сеть, называемую диаграммой Вороного. Диаграмму Вороного множества точек  $S$  будем обозначать  $Vor(S)$ . Вершины многоугольников определяют вершины диаграммы Вороного, а соединяющие их отрезки – ребра диаграммы Вороного. Каждая из  $N$  исходных точек множества принадлежит в точности одному многоугольнику Вороного.

го. Поэтому, если  $(x, y) \in V(i)$ , то  $p_i$  является ближайшим соседом точки  $(x, y)$ . Диаграмма Вороного содержит всю информацию о близости точек соответствующего множества.

**4. Свойства диаграммы Вороного.** Здесь приводятся без доказательств основные свойства диаграммы Вороного. Доказательства можно посмотреть в книге Ф.Препарата, М.Шеймос “Вычислительная геометрия: Введение”. Будем считать справедливым следующее предположение.

**Предположение.** *Никакие четыре точки исходного множества  $S$  не лежат на одной окружности.*

Если отказаться от этого предположения, то в формулировки потребуется внести несущественные, но довольно длинные уточнения.

Для начала заметим, что каждое ребро диаграммы Вороного является отрезком прямой, перпендикулярной отрезку, соединяющему некоторую пару точек множества  $S$ , и делящей этот отрезок пополам. Таким образом, ребро принадлежит в точности двум многоугольникам.

**Теорема 1.** *Каждая вершина диаграммы Вороного является точкой пересечения в точности трех ребер.*

Теорема 1 эквивалентна следующему утверждению: вершины диаграммы Вороного являются центрами окружностей, каждая из которых определяется тремя точками исходного множества, а сама диаграмма Вороного является регулярной со степенью вершин, равной трем. Обозначим через  $C(v)$  упомянутую выше окружность, соответствующую вершине  $v$ .

**Теорема 2.** *Для каждой вершины  $v$  диаграммы Вороного множества  $S$  окружность  $C(v)$  не содержит никаких других точек множества  $S$ .*

**Теорема 3.** *Каждый ближайший сосед точки  $p_i$  в  $S$  определяет ребро в многоугольнике Вороного  $V(i)$ .*

**Теорема 4.** *Многоугольник  $V(i)$  является неограниченным тогда и только тогда, когда точка  $p_i$  лежит на границе выпуклой оболочки множества  $S$ .*

Так как лишь неограниченные многоугольники могут иметь в качестве ребер лучи, то лучи диаграммы Вороного соответствуют парам смежных точек множества  $S$ , лежащих на границе выпуклой оболочки.

Рассмотрим теперь граф, двойственный диаграмме Вороного, то есть граф, расположенный на плоскости и получаемый в результате соединения отрезками каждой пары точек множества  $S$ , многоугольники Вороного которых имеют общее ребро. В результате получается граф с вершинами в исходных  $N$  точках. На первый взгляд граф, двойственный диаграмме Вороного, может показаться довольно необычным, так как ребро диаграммы может даже не пересекаться с двойственным ему ребром. Важность двойственного графа во многом обусловлена следующей теоремой Делоне:

**Теорема 5.** Граф, двойственный диаграмме Вороного, является триангуляцией множества  $S$ .

**Следствие.** Диаграмма Вороного множества из  $N$  точек имеет не более  $2N - 5$  вершин и  $3N - 6$  ребер.

Так как диаграмма Вороного является планарным графом, то для ее представления достаточно линейного по числу точек объема памяти. Это позволяет представить информацию о близости в чрезвычайно компактной форме. Любой конкретный многоугольник Вороного может иметь до  $N - 1$  ребер, но полное число ребер не превосходит  $3N - 6$ , при этом каждое ребро принадлежит в точности двум многоугольникам. Это значит, что среднее число ребер многоугольника Вороного не превосходит шести.

**5. Построение диаграммы Вороного.** Под задачей построения диаграммы Вороного  $Vor(S)$  на множестве точек  $S$  будем понимать порождение описания диаграммы как планарного графа, расположенного на плоскости. Это описание должно включать координаты вершин диаграммы Вороного и множество ребер диаграммы, каждое из которых представляется парой вершин диаграммы, представленное в форме РСДС.

Теперь рассмотрим сначала вопрос о нижних оценках для времени, необходимого для построения диаграммы Вороного.

**Теорема 6.** В рамках модели алгебраических деревьев вычислений для построения диаграммы Вороного множества из  $N$  точек требуется  $O(N \log N)$  операций в худшем случае.

Простой (но достаточно грубый) подход к построению диаграммы Вороного заключается в поочередном построении многоугольников, входящих в диаграмму. Так как каждый многоугольник Вороного получается в результате пересечения  $N - 1$  полуплоскостей, то, используя метод, который был рассмотрен в лекции 2008 года, можно построить этот многоугольник за время  $O(N \log N)$ , что дает для полного времени построения всех многоугольников оценку  $O(N^2 \log N)$ . Далее будет показано, что вся диаграмма целиком может быть построена за оптимальное время  $O(N \log N)$ , а это значит, что асимптотически задача построения всей диаграммы является не более сложной, чем задача построения лишь одного из многоугольников этой диаграммы!

Несмотря на кажущуюся сложность, задача построения диаграммы Вороного прекрасно подходит для применения метода “разделяй и властвуй”. Успех применяемого метода зависит от различных структурных свойств диаграммы, использование которых позволяет произвести объединение решений подзадач за линейное время. Попытаемся в общих чертах описать алгоритм:

Шаг 1 Разделить множество  $S$  на два приблизительно равных подмн-

жества  $S_1$  и  $S_2$ .

Шаг 2 Рекурсивно построить  $Vor(S_1)$  и  $Vor(S_2)$ .

Шаг 3 Объединить  $Vor(S_1)$  и  $Vor(S_2)$  и таким образом получить  $Vor(S)$ .

Предположим, что шаг 1 алгоритма можно выполнить за время  $O(N)$ . Если обозначить через  $T(N)$  полное время работы алгоритма, то на выполнение шага 2 потребуется время, равное примерно  $2T(N/2)$ . Таким образом, если удастся объединить  $Vor(S_1)$  и  $Vor(S_2)$  за линейное время, получив в результате диаграмму Вороного  $Vor(S)$  всего исходного множества, то мы получим оптимальный алгоритм с временем работы  $O(N \log N)$ . Но прежде чем браться за более детальную разработку алгоритма, попытаемся ответить на следующий вопрос: на каком основании можно считать, что  $Vor(S_1)$  и  $Vor(S_2)$  как-то связаны с  $Vor(S)$ ?

Чтобы ответить на этот вопрос, определим геометрическую конструкцию, играющую ключевую роль в рассматриваемом подходе.

**Определение.** Для заданного разбиения  $\{S_1, S_2\}$  множества  $S$  обозначим  $\sigma(S_1, S_2)$  – множество ребер диаграммы Вороного, общих для пар многоугольников  $V(i)$  и  $V(j)$  диаграммы  $Vor(S)$ , где  $p_i \in S_1$ ,  $p_j \in S_2$ .

Совокупность ребер  $\sigma(S_1, S_2)$  обладает следующими свойствами.

**Теорема 7.** Совокупность  $\sigma(S_1, S_2)$  является множеством ребер некоторого подграфа диаграммы  $Vor(S)$  и обладает следующими свойствами:

1.  $\sigma(S_1, S_2)$  состоит из циклов и цепей, не имеющих общих ребер. Если некоторая цепь содержит лишь одно ребро, то это ребро является прямой линией; иначе два крайних ребра цепи являются лучами.
2. Если множества  $S_1$  и  $S_2$  линейно разделимы, то  $\sigma(S_1, S_2)$  состоит из единственной монотонной цепи.

В том случае, когда множества  $S_1$  и  $S_2$  являются линейно разделимыми, единственную цепь, содержащуюся в  $\sigma(S_1, S_2)$ , будем обозначать  $\sigma$ . Если отделяющая прямая  $t$  является вертикальной, то можно недвусмысленно говорить о том, что  $\sigma$  разбивает плоскость на левую  $\pi_L$  и правую  $\pi_R$  части. Имеет место следующее важное свойство.

**Теорема 8.** Если множества  $S_1$  и  $S_2$  линейно разделимы вертикальной прямой  $t$ , при этом  $S_1$  находится слева от  $S_2$ , то диаграмма Вороного  $Vor(S)$  представляет собой объединение  $Vor(S_1) \cap \pi_L$ ,  $Vor(S_2) \cap \pi_R$  и  $\sigma$ .

Теперь мы можем уточнить алгоритм построения диаграммы.

Шаг 1 Разделить множество  $S$  на два приблизительно равных подмножества  $S_1$  и  $S_2$ , использовав для этого медиану по координате  $x$ .

Шаг 2 Рекурсивно построить  $Vor(S_1)$  и  $Vor(S_2)$ .

Шаг 3' Построить ломаную  $\sigma$ , разделяющую  $S_1$  и  $S_2$ .

Шаг 3" Удалить все ребра диаграммы  $Vor(S_2)$ , расположенные слева от  $\sigma$ , и все ребра  $Vor(S_1)$ , расположенные справа от  $\sigma$ . В результате получаем  $Vor(S)$  – диаграмму Вороного для множества в целом.

Очевидно, что успешное выполнение этой процедуры зависит от того, насколько быстро мы сможем построить  $\sigma$ , так как шаг 3" не вызывает затруднений. Если говорить о времени обработки, то начальное разбиение множества  $S$  с использованием медианы по координате  $x$  может быть выполнено за  $O(N)$  с помощью стандартного алгоритма поиска медианы. Шаг 3" может быть выполнен за время  $O(|S_1| + |S_2|) = O(N)$ . Остается лишь найти эффективный способ построения разделяющей цепи  $\sigma$ .

**6. Построение разделяющей цепи.** Первый шаг в построении разделяющей цепи  $\sigma$  заключается в определении двух ее лучей. Заметим, что каждый луч цепи  $\sigma$  перпендикулярен опорному отрезку (каждый своему) к  $CH(S_1)$  и  $CH(S_2)$  и делит его пополам. Отметим также, что так как по предположению  $S_1$  и  $S_2$  линейно разделимы, то имеются в точности два опорных отрезка к их оболочкам (тем самым подтверждается, что  $\sigma(S_1, S_2)$  состоит лишь из одной цепи). Если теперь (по индукции) предположить, что уже построены эти две выпуклые оболочки, то два опорных отрезка, которые мы обозначим  $t_1$  и  $t_2$ , строятся (самое большее) за линейное время, после чего легко определяются лучи цепи  $\sigma$ . Заметим, что в качестве побочного результата мы получаем также  $CH(S)$ , обеспечивая тем самым наличие выпуклых оболочек, необходимых для шага индукции. Найдя луч цепи  $\sigma$ , мы последовательно строим ребра цепи до тех пор, пока не достигнем второго луча. Как только цепь доходит до одного из ребер диаграммы  $S_1$  или  $S_2$ , она изменяет свое направление, поскольку изменяется многоугольник, которому она принадлежит в одной из диаграмм.

Предположим, что продвижение по  $\sigma$  осуществляется в направлении уменьшения координаты  $y$ . Механизм продвижения по  $\sigma$  обеспечивает переход от текущего ребра  $e$  и текущей вершины  $v$  (являющейся верхним концом ребра  $e$ ) к следующим ребру  $e'$  и вершине  $v'$ . Если ребро  $e$  разделяет многоугольники  $V(i)$  и  $V(j)$ , где  $p_i \in S_1$  и  $p_j \in S_2$ , то вершина  $v'$  является либо точкой пересечения  $e$  с границей  $V(i)$  в  $Vor(S_1)$ , либо точкой пересечения  $e$  с границей  $V(j)$  в  $Vor(S_2)$  в зависимости от того, какая из них ближе к  $v$ . Таким образом, просмотрев границы многоугольников  $V(i)$  в  $Vor(S_1)$  и  $V(j)$  в  $Vor(S_2)$ , можно легко определить

вершину  $v'$ . К сожалению, цепь  $\sigma$  может, оставаясь внутри некоторого многоугольника  $V(i)$ , многократно менять свое направление, прежде чем она пересечет какое-либо ребро многоугольника  $V(i)$ . И если каждый раз приходится просматривать все многоугольники  $V(i)$ , то может потребоваться проверить слишком много ребер, и в общем случае необходимое для этого время может оказаться квадратичным. Однако использование особенностей структуры диаграммы Вороного позволяет разработать более эффективную стратегию просмотра.

Действительно, предположим, что последовательность ребер  $e_1, e_2, \dots, e_k$  цепи  $\sigma$  содержится в  $V(i)$ , где для конкретности  $p_i \in S_1$ . Продолжим каждое ребро  $e_h$  за его конечную точку  $v_h$ , получая луч  $r_h$ , и рассмотрим точку пересечения  $q_h$  луча  $r_h$  с границей многоугольника  $V(i)$  в  $Vor(S_1)$ . Рассмотрим подцепь  $C_1 = e_1, e_2, \dots, e_h$  и часть границы многоугольника  $V(i)$  в  $Vor(S_1)$ , расположенную справа от  $\sigma$ , которую обозначим  $C_2$ . Подцепь  $C_2$  – это часть границы (возможно, неограниченного) выпуклого многоугольника  $V(i)$  в  $Vor(S_1)$ ;  $C_1$  также является выпуклой и целиком содержится в  $V(i)$ . Так как все углы между отрезками  $v_h q_h$  и  $v_{h+1} q_{h+1}$ , где  $h = 1, 2, \dots, k - 1$  (с учетом направления) имеют один и тот же знак, то точки пересечения  $q_1, q_2, \dots, q_k$  оказываются упорядоченными на  $C_2$ . Отсюда следует, что при определении точек пересечения  $q$  цепь  $C_2$ , то есть границу многоугольника  $V(i)$  в  $Vor(S_1)$ , необходимо просматривать без возвратов в порядке обхода по часовой стрелке. Аналогичные рассуждения показывают, что границу любого многоугольника  $V(j)$  в  $Vor(S_2)$  следует просматривать без возвратов в порядке обхода против часовой стрелки.

Теперь можем записать алгоритм для построения разделяющей цепи. В работе алгоритма используются три ребра: два ребра  $e_L$  и  $e_R$ , принадлежащие соответственно  $Vor(S_1)$  и  $Vor(S_2)$ , и “текущее ребро”  $e$  цепи  $\sigma$  (в действительности не само ребро, а прямая, содержащая это ребро). Кроме ребра  $e$  используется также его начальная точка  $v$  (для первого ребра  $e^*$  цепи  $\sigma$ , являющегося лучом, в качестве  $v^*$  берется точка, лежащая на  $e^*$  и имеющая достаточно большую ординату). И, наконец, используются две точки множества  $S$ ,  $p_L \in S_1$ ,  $p_R \in S_2$ , образующие отрезок  $p_L p_R$ , перпендикулярный текущему ребру  $e$ , которое делит его пополам. Пусть  $I(e, e')$  обозначает пересечение отрезков  $e$  и  $e'$ , а запись  $I(e, e') = \Lambda$ , что отрезки  $e$  и  $e'$  не пересекаются. Как и ранее,  $t_1$  и  $t_2$  – два опорных отрезка и при этом  $t_1 = pq$ .

```

begin
     $p_L := p$ 
     $p_R := q$ 
     $e := e^*$ 
     $e_L :=$  первое ребро (открытой) границы  $V(p_L)$ 
     $e_R :=$  первое ребро (открытой) границы  $V(p_R)$ 
    while  $p_L p_R \neq t_2$  do
        while  $I(e, e_L) = \Lambda$  do
            |  $e_L := P_2[e_L]$ 
        end
        while  $I(e, e_L) = \Lambda$  do
            |  $e_R := R[P_1[R[e_R]]]$ 
        end
        if  $v$  ближе к  $I(e, e_L)$ , чем к  $I(e, e_R)$  then
            |  $v := I(e, e_L)$ 
            |  $e_L := R[e_L]$ 
            |  $p_L := F[e_L]$ 
            |  $e :=$  прямая, перпендикулярная  $p_L p_R$  и делящая его пополам
        end
        else
            |  $v := I(e, e_R)$ 
            |  $e_R := R[e_R]$ 
            |  $p_R := F[e_R]$ 
            |  $e :=$  прямая, перпендикулярная  $p_L p_R$  и делящая его пополам
        end
    end
end

```

## Задачи и разборы

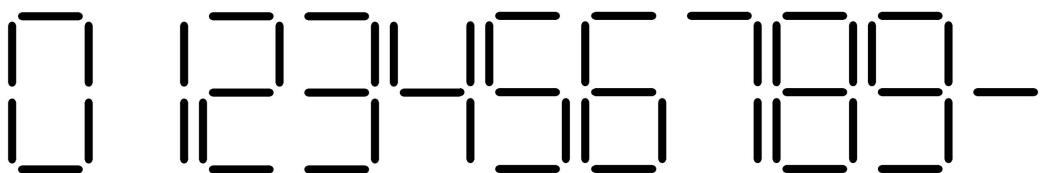
### Задача А. Количество линий

Имя входного файла:	<code>a.in</code>
Имя выходного файла:	<code>a.out</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Известный программист Петя планирует создать совершенно новую увлекательную компьютерную игру, которая не снилась никому из ведущих фирм, занимающихся разработкой игр. Петя уверен, что благодаря тем революционным особенностям, которые он намерен реализовать, его игра

быстро завоюет рынок и принесет ему всемирную славу и баснословные прибыли.

Одной из таких особенностей будет система отображения результатов в таблице рекордов. Каждый результат – это число, записываемое в десятичной системе. Но цифры и знак минус будут отображаться с помощью нескольких линий:



Помогите Пете узнать сколько линий его программа нарисует, когда игрок наберет определенное количество очков.

### **Формат входного файла**

В единственной строке задается одно целое число  $X$  ( $-10^9 \leq X \leq 10^9$ ), определяющее количество очков, набранных игроком.

### **Формат выходного файла**

В единственную строку выведите одно число – количество линий, необходимых для изображения результата.

### **Примеры**

<b>a.in</b>	<b>a.out</b>
85	12
-3	6

### **Разбор задачи А. Количество линий**

Задача очень простая. Требуется внимательно подсчитать вручную для каждой цифры от 0 до 9 количество линий, из которых состоит ее изображение, и записать в массив. В программе потребуется лишь разбить число на цифры, для каждой из них взять соответствующее значение из массива и сложить все эти значения. Отдельное внимание требуют лишь число 0 и отрицательные числа (у которых есть знак –, состоящий из одной линии). Наиболее простой способ получения цифр – это читать из файла не число, а его цифры посимвольно. В этом случае проблемы с числом 0 и отрицательными числами не возникает вовсе.

## Задача В. Восстановление количества очков

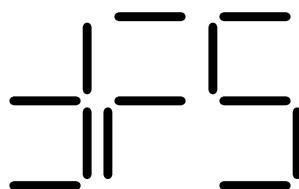
Имя входного файла: **b.in**

Имя выходного файла: **b.out**

Ограничение по времени: 1 с

Ограничение по памяти: 64 Мб

Не менее известный хакер Вася, увидев замечательную игру Пети, решил немного испортить ее. В подпрограмме вывода количества очков участника он подправил код таким образом, что каждая из линий, которая используется в изображении, может быть выведена на экран, а может быть и пропущена. Таким образом, игрок не всегда мог ясно понять сколько у него очков. Например, число 325 могло бы быть изображено как



Но ведь таким же образом могло быть записано и число 986, и еще несколько чисел. Ваша задача – по заданному изображению определить сколько целых чисел, определяющих количество очков, могло быть записано таким образом.

### Формат входного файла

В первой строке задается число  $K$  ( $1 \leq K \leq 9$ ) – количество символов в числе, которое нужно было отобразить. В каждой из следующих  $K$  строк записаны по 7 цифр, каждая из которых либо 0, либо 1. Эти цифры соответствуют линиям в изображении соответствующего символа числа, перечисленным сверху вниз, а на одном уровне слева направо. То есть первая цифра соответствует самой верхней линии, вторая – верхней левой, третьей – верхней правой, четвертая – средней, пятая – нижней левой, шестая – нижней правой, седьмая – самой нижней. Значение 1 обозначает наличие линии, а 0 – отсутствие. Изображение задается слева направо, то есть первым дается описание изображения самого левого символа, использованного в записи числа, последним – самого правого.

### Формат выходного файла

В единственную строку выведите одно число – количество чисел, которые могли быть изображены заданным образом. Учтите, что при выводе какого-либо символа могло быть не отображено ни одной линии, однако

при правильном выводе ни одно знакоместо не должно было оказаться пустым.

## Примеры

b.in	b.out
3 0011011 1001100 1101011	36
1 1110111	2
2 0000000 0000000	99

## Разбор задачи В. Восстановление количества очков

В отличие от предыдущей задачи, здесь уже потребуется гораздо больше подготовительной работы. Так или иначе прийдется внести в программу информацию о том, какие линии входят в изображение каждой цифры. Можно это сделать, например, с помощью матрицы, в которой элемент в позиции  $(i, j)$  будет равен 1 если линия с номером  $j$  есть в изображении цифры  $i$ . Другой вариант – закодировать изображение каждой цифры в виде строки в формате, описанном во входных данных. Но наиболее удобным будет представление в виде двоичной маски. Для каждой цифры мы запишем число, двоичное представление которого совпадает со строкой, задающей изображение этой цифры. Так, например, изображение цифры 4 кодируется строкой 0111010, которое может быть представлено в программе как `0x3A` в шестнадцатеричной форме.

Теперь нам потребуется уметь определять, могут ли какие-то цифры быть получены из заданного изображения. Изображение мы точно так же переводим в двоичное число  $x$ . Это изображение может соответствовать цифре с маской  $mask$ , если биты, равные 1 в  $x$ , равны также 1 и в  $mask$ . Поэтому, если мы выполним операцию `and` над числами  $x$  и  $mask$ , должно получиться то же самое число  $x$ . Либо можно промаскировать  $x$  обратной к  $mask$  маской, в этом случае должен получиться 0.

Теперь задача сводится к обычной комбинаторной, к которой можно применить основные правила комбинаторики – сложения и умножения.

*Правило сложения.* Если все множество вариантов  $A$  можно разбить на два непересекающихся подмножества  $A_1$  и  $A_2$ , причем выбрать элемент из

$A_1$  можно  $n_1$  способами, а из  $A_2 - n_2$  способами, то из множества  $A$  можно выбрать вариант  $n_1 + n_2$  способами.

*Правило умножения.* Если выбор варианта из множества  $A$  сводится к тому, что необходимо выбрать один элемент из множества  $A_1$  и один элемент из множества  $A_2$  независимо от первого выбора, причем выбрать элемент из  $A_1$  можно  $n_1$  способами, а из  $A_2 - n_2$  способами, то из множества  $A$  можно выбрать вариант  $n_1 \cdot n_2$  способами.

Эти правила легко обобщаются и на большее количество множеств.

Как следует из предыдущей задачи, количество очков в игре может быть как положительным, так и отрицательным, и даже нулевым. Разобьем все множество чисел на 3 указанных подмножества. Тогда согласно правилу сложения ответ будет определяться суммой количеств вариантов для каждого из этих подмножеств. Эти количества будет легко определить с помощью правила умножения, определив сколько есть вариантов заполнения цифрой для каждого знакоместа. Отметим, что в положительном числе в первом знакоместе может находиться любая цифра, кроме 0, для остальных знакомест ограничений нет. Число 0 должно быть записано ровно одной цифрой, поэтому если  $K > 2$ , то вариант с нулевым количеством очков невозможен. Наконец, для отрицательного числа должно быть по крайней мере два знакоместа, в первом из которых должен находиться знак  $-$ , во втором любая цифра отличная от 0, остальные же знакоместа могут быть заполнены произвольными цифрами.

### Задача С. Как побить все рекорды

Имя входного файла:	c.in
Имя выходного файла:	c.out
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Кроме известных программистов и хакеров, есть также и широко известные геймеры. Коля как раз и есть один из них. Он очень любит играть в игры и ставить в них рекорды.

Однажды ему попалась в руки игра Пети и он решил поставить в ней такой рекорд, который никому и никогда не удастся побить. Очевидно для этого надо набрать максимально возможное число очков. Коля знает, что в начале игры у игрока 0 очков. И каждый ход он может заработать от  $a$  до  $b$  очков включительно (не исключаются и отрицательные числа – они означают, что игрок штрафуется на некоторое количество очков). При этом количество ходов никак не ограничено, но игру можно закончить в любой удобный момент.

Кроме того, хакер Вася сообщил Коле по секрету, что для хранения количества очков в программе Пети использована переменная  $n$ -байтового целочисленного типа со знаком. Поэтому количество очков может принимать любое целое значение от  $-2^{8n-1}$  до  $2^{8n-1} - 1$ . Переменные такого типа обладают тем свойством, что если к максимальному значению ( $2^{8n-1} - 1$ ) прибавить 1, то произойдет переполнение и в результате получится минимальное ( $-2^{8n-1}$ ). Верно и обратное – если из минимального значения вычесть единицу (или, что то же самое, добавить  $-1$ ) получится максимальное. Добавление любого положительного числа  $k$  означает  $k$ -кратное применение операции увеличения на единицу. Аналогично, добавление отрицательного числа означает применение соответствующего количества раз операции уменьшения на 1.

Помогите Коле определить минимальное количество ходов, которые потребуется ему, чтобы набрать максимально представимое количество очков.

### Формат входного файла

В единственной строке задаются три целых числа  $n$ ,  $a$  и  $b$  ( $1 \leq n \leq 8$ ,  $-2^{8n-1} \leq a \leq 0 \leq b \leq 2^{8n-1} - 1$ ).

### Формат выходного файла

В единственную строку выведите одно число – количество ходов для установления рекорда, равного максимально представимому количеству очков. Если это сделать невозможно, выведите число  $-1$ .

### Примеры

c.in	c.out
1 0 8	16
2 -1 151	217
4 -1000000 0	2148

### Разбор задачи С. Как побить все рекорды

Для решения задачи важно понять, что набрать максимальное количество очков можно двумя способами – либо набрать действительно максимальное количество очков, либо на один меньше минимального. Если  $b \neq 0$ , то максимальное число очков можно набрать за  $\lceil \frac{MAX}{b} \rceil$  ходов. Если  $a \neq 0$ , то на один меньше минимума можно набрать за  $\lceil \frac{-(MIN-1)}{-a} \rceil$  ходов. Остается выбрать только из этих величин наименьшую. Невозможно получить максимального количества очков лишь в том случае, когда  $a = b = 0$ .

В этой задаче следует очень внимательно работать с целочисленными типами, поскольку при  $n = 8$  значение  $MIN = -2^{63}$  может быть представлено лишь в типе `long long`, а значение  $-MIN$  – лишь в типе `unsigned long long`.

## Задача D. Секретный уровень

Имя входного файла:	<code>d.in</code>
Имя выходного файла:	<code>d.out</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Поиграв достаточное время в игру Пети, Коля обнаружил, что в ней есть секретный уровень, для получения доступа к которому нужно ввести очень секретный код и очень долго ждать проверки его правильности. Коля даже однажды видел и запомнил код, который вводил Петя, играя в свою игру, но, попытавшись ввести самостоятельно этот код, получил сообщение о том, что он неверный. Поскольку уровень обещал быть невероятно богат разными плюшками, Коля решил, что должен непременно на него попасть и обратился за помощью к хакеру Васе. После долгого изучения кода игры Пети Васе так и не удалось снять защиту с этого уровня, но удалось выяснить как можно получить этот код.

Способ получения кода заключается в следующем. Берется число, равное серийному номеру компьютера, на котором запущена игра. Далее каждая цифра этого числа умножается на саму себя, полученные при этом числа записываются рядом друг с другом, образуя новое число. Эта операция выполняется столько раз, сколько было перезагрузок компьютера в течение дня. Кодом будет количество цифр в числе, которое получится в итоге.

### Формат входного файла

В единственной строке задаются два целых числа: количество перезагрузок  $k$  и серийный номер компьютера Коли  $N$  ( $0 \leq k \leq 100$ ,  $0 \leq N \leq 10^{13}$ ).

### Формат выходного файла

В единственную строку выведите одно число – код для секретного уровня в игре Пети.

## Примеры

<b>d.in</b>	<b>d.out</b>
1 85	4
2 4	3

## Примечание

В первом примере, после возвведения в квадрат каждой цифры, получается число 6425, состоящее из четырех цифр. Во втором примере, сначала 4 возводится в квадрат – получается 16, а после возведения в квадрат цифр числа 16 получится трехзначное число 136.

## Разбор задачи D. Секретный уровень

Числа, получающиеся в результате преобразований, могут оказаться весьма большими. Поэтому, непосредственное моделирование всего процесса потребует как очень большого объема памяти, так и огромного количества времени. Поскольку требуется получить не само число, а количество цифр в нем, то информации о том, сколько раз встречается каждая цифра в очередном полученном числе будет достаточно для решения задачи. Остается лишь правильно построить преобразование. Так, например, количество единиц  $k'_1$  после преобразования может быть вычислено как  $k_1 + k_4 + k_9$ , где  $k_i$  – количество цифр  $i$  до преобразования. При таком подходе конечно малейшая ошибка в формуле приведет к неверной программе. Более устойчивым будет вычисление квадрата для каждой цифры  $i$  и увеличении  $k'_{i^2 \bmod 10}$  и  $k'_{[i^2/10]}$  (при  $i^2 \leq 10$ ) на  $k_i$ .

Следует отметить, что длина числа получающегося после преобразований может уместиться лишь в 64-разрядном целочисленном типе. Кроме того, нужно аккуратно обрабатывать случаи  $k = 0$  (когда не выполняется ни одного преобразования) и  $N = 0$  (при разбиении числа на цифры помнить, что в этом числе есть одна цифра 0).

## Задача E. Поля сражений

Имя входного файла: e.in  
Имя выходного файла: e.out  
Ограничение по времени: 1 с  
Ограничение по памяти: 64 Мб

В своей игре Петя предполагает сделать битвы между армиями противников на прямоугольных полях, разбитых на квадратные клетки. Такие

поля есть во многих играх, однако задумка Пети заключается в том, что каждое поле сражения будет состоять из вполне определенного количества клеток. Каждое следующее сражение будет происходить на поле, содержащем на одну клетку больше, чем предыдущее. Длина и ширина полей значения не имеют, их можно выбрать как угодно. Однако поля размера  $1 \times k$  Петя считает слишком простыми и не хочет, чтобы они использовались в его игре.

Известно, что в игре состоится  $N$  сражений. Помогите Пете выбрать количество клеток на самом первом поле, так чтобы из этого и всех последующих  $N - 1$  числа клеток могло быть составлено хотя бы по одному непростому полу.

### **Формат входного файла**

В единственной строке задано одно целое число  $N$  ( $1 \leq N \leq 10000$ ).

### **Формат выходного файла**

В единственную строку выведите целое число – количество клеток на первом из  $N$  последовательных непростых полей. Это число не обязано быть минимальным, однако не должно превышать  $10^{4500}$ . Если числа с указанными свойствами не существует, выведите значение 0.

### **Примеры**

<b>e.in</b>	<b>e.out</b>
1	4
2	8
3	14
4	24
5	32

### **Разбор задачи Е. Поля сражений**

По-видимому, это самая сложная задача юниорской лиги. Конечно, достаточно найти всего одно составное число, за которым будут следовать еще 9999 составных чисел. И такое число будет удовлетворять условию задачи при любых допустимых  $N$ , однако найти такое число отнюдь не проще, чем решить задачу для произвольного  $N$ .

Предположим, что есть некоторое число  $A$ , которое делится на  $N$  последовательных натуральных чисел от  $k$  до  $k + N - 1$ , где  $k > 1$ . Тогда числа от  $A + k$  до  $A + k + N - 1$  будут составными. Действительно, число  $A + i$ , где  $k \leq i \leq k + N - 1$ , будет тогда кратным  $i$ . Ясно, что число

$A = k \cdot (k+1) \cdot \dots \cdot (k+N-1)$  удовлетворяет нашим требованиям, а значит значение  $A + k$  будет решением задачи. Однако, даже если взять минимальное допустимое  $k$ , то есть  $k = 2$ , то полученное число  $(N+1)! + 2$  будет при больших  $N$  иметь длину превышающую указанное в условии значение. Попробуем его уменьшить. Дело в том, что  $(N+1)!$  далеко не минимальное значение кратное всем числам от 2 до  $N+1$ . Таковым будет число НОК( $2, 3, \dots, N+1$ ). Но даже это значение при  $N$  близких к 10000 будет иметь порядка 5000–6000 десятичных знаков.

Заметим, что совсем необязательно, чтобы число  $A + i$  делилось именно на  $i$ . Достаточно лишь, чтобы не оно было простым, например, чтобы  $A$  и  $i$  имели некоторый общий простой делитель. Поскольку все простые делители  $i$  не превосходят его самого, то число  $A$  может быть выбрано таким образом, чтобы оно делилось на все простые числа, не превосходящие  $N+1$ , то есть  $A = p_1 p_2 \dots p_{\pi(N+1)}$ , где  $p_j$  –  $j$ -ое по порядку простое число. Такое значение  $A$  будет удовлетворять ограничениям задачи даже при максимальном значении  $N$ .

### Задача F. Взлом таблицы рекордов

Имя входного файла:	<code>f.in</code>
Имя выходного файла:	<code>f.out</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Хакеру Васе не дает покоя то, что первое место в таблице рекордов Петиной игры занимает какой-то Коля. Хитроумным приемом ему удалось добиться того, чтобы в таблице результатов напротив его имени стояло число, задаваемое машинным кодом игры. Однако, чтобы рекорд выглядел более-менее правдоподобным, он решил вычеркнуть из получившегося числа определенное количество цифр, но так, чтобы оставшееся число было как можно больше.

Помогите Васе узнать максимальное число, которое может получиться после вычеркивания.

#### Формат входного файла

В первой строке задаются два целых числа  $N$  и  $k$  ( $0 \leq k < N \leq 300000$ ), количество цифр в исходном числе и количество цифр, которое должно быть вычеркнуто. Вторая строка содержит исходное натуральное число, состоящее из  $N$  цифр.

## Формат выходного файла

В единственную строку выведите максимальное число, которое может получиться у Васи.

## Примеры

<b>f.in</b>	<b>f.out</b>
7 6 1234321	4
3 1 718	78
4 2 1224	24

## Разбор задачи F. Взлом таблицы рекордов

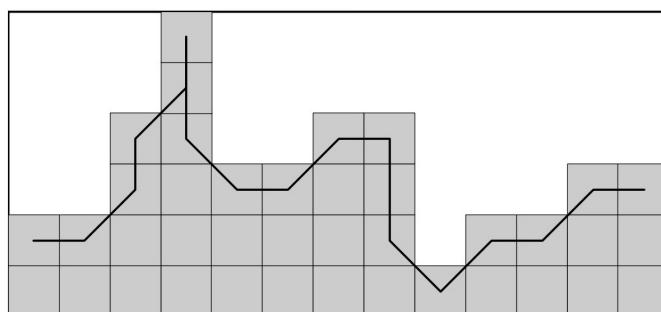
Ясно, что после вычеркивания из  $N$ -значного числа  $k$  цифр, получится  $(N - k)$ -значное число. Если взять два числа, которые получаются, то их можно сравнивать лексикографически. То есть большим из них будет то, у которого больше будет первая цифра, при их равенстве – у кого больше следующая цифра, и т.д. Ясно, что на место первой цифры могут претендовать лишь те, что стоят в исходном числе на позициях с 1 по  $N - k + 1$ . Из них очевидно следует выбрать наибольшую, находящуюся как можно ближе к началу числа (такой вариант заведомо не хуже любого другого с наибольшей цифрой, поскольку заменить первую цифру на такую же, но стоящую раньше, всегда возможно, а вот наоборот на ту, которая стоит дальше – не всегда). Допустим, выбрана была позиция  $i$ , тогда следующей цифрой может быть цифра, стоящая на позициях с  $i + 1$  по  $N - k + 2$ . Продолжая нахождение самого раннего максимума дальше, получим наибольший возможный результат. Однако непосредственная реализация указанного алгоритма потребует выполнения порядка  $O(N^2)$  операций. Для того, чтобы оптимизировать этот подход, будем искать эти  $N - k$  максимумов, одновременно. Допустим, у нас после просмотра  $i - 1$  цифры найдено  $j$  первых цифр  $m_1, \dots, m_j$ . Возьмем очередную цифру  $a_i$ , она может претендовать на то, чтобы стать в массиве  $m$  на позиции с  $\max(j - N + k, 1)$  по  $j + 1$ . Этот диапазон всегда будет содержать либо только одну позицию  $j + 1$ , либо значения  $m$  в этих позициях будут упорядочены по убыванию. Поэтому с помощью двоичного поиска можно найти позицию  $l$  в этом диапазоне такую, что  $m_{l-1} \geq a_i > m_l$ . Именно на позицию  $l$  мы и поставим  $a_i$ , принимая в дальнейшем, что  $j = l$  (ясно, что теперь цифры, которые в результате идут за  $l$ -ой, нужно заново находить). Такой алгоритм потребует

уже только  $O(N \log(N - k))$  действий и поэтому уложится в отведенный лимит времени. Но можно еще улучшить этот алгоритм, если реализовать  $t$  в виде стека. Будем подсчитывать, сколько элементов у нас уже удалялось из стека. Если стек пуст, или количество удаленных элементов уже достигло  $k$ , или на вершине стека лежит число не меньшее очередной цифры  $a_i$ , то кладем  $a_i$  в стек и переходим к следующей цифре исходного числа. В противном случае, мы убираем элементы с вершины стека до тех пор, пока предыдущее условие не станет верным. Если после того, как все число было обработано, количество удаленных элементов оказалось меньше  $k$ , то нужно принудительно довести его до  $k$ , снимая элементы с вершины стека. Результат получится, если вывести значения от дна стека до его вершины. Нетрудно видеть, что результат работы такого алгоритма будет в точности совпадать с тем, что получалось и в алгоритме, использующем бинарный поиск. Но время работы этого алгоритма будет уже  $O(N)$ , поскольку каждый элемент ровно один раз помещается в стек и не более одного раза удаляется из него, а на каждом шаге алгоритма выполняется либо одна, либо другая операция. Для упрощения реализации последнего алгоритма можно ввести дополнительные фиктивные элементы в начало и в конец массива цифр исходного числа  $a_0 = +\infty$ ,  $a_{N+1} = -\infty$ .

### Задача G. Горная гряда

Имя входного файла:	g.in
Имя выходного файла:	g.out
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Во время сражений в игре Пети на дальнем плане изображается горная гряда, которая представляет собой связное и выпуклое в вертикальном направлении множество пикселей, нижняя сторона которого параллельна горизонтали.



Поскольку, как сказал Вася, эта гряда генерируется с помощью датчика случайных чисел, который потом используется и в процессе боя, Коля

уверен, что изучив внимательно горы, он сможет предугадывать ход боя. В частности, его интересуют протяженность горной гряды слева направо (количество пикселей на нижней стороне) и максимальный перепад высот (расстояние по вертикали между самым верхним и самым нижним пикслем).

### **Формат входного файла**

Гряда задается своей верхней огибающей ломаной, каждое звено которой соединяет соседние пиксели по горизонтали, вертикали или диагонали. В первой строке задается целое число  $N$  ( $1 \leq N \leq 100000$ ), количество звеньев ломаной. В каждой из последующих  $N$  строк задается пара чисел  $x$  и  $y$  – координаты вектора, определяющего соответствующее звено ломаной ( $0 \leq x \leq 1$ ,  $-1 \leq y \leq 1$ ,  $x^2 + y^2 \neq 0$ ).

### **Формат выходного файла**

В единственную строку выведите два целых числа – протяженность горной гряды и максимальный перепад высот.

### **Пример**

<b>g.in</b>	<b>g.out</b>
18 1 0 1 1 0 1 1 1 0 1 0 -1 0 -1 1 -1 1 0 1 1 1 0 0 -1 0 -1 1 -1 1 1 1 0 1 1 1 0	13 5

## Разбор задачи G. Горная гряда

Пусть  $(X_i, Y_i)$  – последовательность координат пикселей вдоль ломаной. ( $i = \overline{0, N}$ ). Значения  $X_0$  и  $Y_0$  можно выбрать совершенно произвольно – от их выбора никак не будет зависеть ответ. Последующие же значения получаются по формулам  $X_i = X_{i-1} + x_i$ ,  $Y_i = Y_{i-1} + y_i$ . Тогда величины, которые требуется найти в задаче, – протяженность  $X_N - X_0 + 1$  (поскольку  $x_i \geq 0$ , то последовательность  $X_i$  монотонно неубывает, и ее экстремальные значения должны быть на концах промежутка), высота  $\max Y_i - \min Y_i$ . Все эти величины легко находятся с помощью стандартных алгоритмов.

## Задача H. Постройка склада

Имя входного файла:	<code>h.in</code>
Имя выходного файла:	<code>h.out</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

На одной из карт размера  $M \times N$  игроку дается герой, который может в одной из клеток карты построить склад, где он будет накапливать различные предметы, которые разбросаны по карте. За один ход герой может:

- переместиться в одну из клеток, соседних по горизонтали или вертикали;
- взять один предмет, если в клетке, где он сейчас находится, есть по крайней мере один предмет, и у него в инвентаре не было ничего (предметы настолько тяжелы, что герой может нести не более одного);
- передать предмет, имеющийся в инвентаре, в склад, если герой находится в клетке, где был построен склад.

Коля хочет знать в каком месте карты следует строить склад, чтобы после его постройки требовалось бы минимальное число ходов на нахождение и складирование всех предметов, которые есть на карте.

### Формат входного файла

В первой строке заданы целые числа  $M$  и  $N$ , определяющие размеры карты ( $1 \leq M, N \leq 1000$ ). В последующих  $M$  строках записано по  $N$  целых чисел, определяющих количества предметов в соответствующих клетках карты. Все эти числа неотрицательные и не превосходят 1000.

## Формат выходного файла

В первой строке выведите координаты (номер строки и столбца) клетки, в которой следует разместить склад, а во второй – количество ходов, которое потребуется после этого для того, чтобы собрать все предметы. Учтите, что если предмет лежит в той же клетке, где уже находится склад, все равно его следует поднять и переложить в склад.

## Примеры

<b>h.in</b>	<b>h.out</b>
4 4 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0	2 3 16
4 3 0 0 2 0 1 1 0 1 0 2 0 0	2 2 38

## Разбор задачи Н. Постройка склада

Рассмотрим сначала одномерный вариант этой задачи. Пусть имеется полоса, состоящая из  $N$  клеток, и  $a_i$  – количество предметов в клетке  $i$ . Предположим, склад строится в клетке с номером  $k$ . Тогда количество ходов будет выражаться формулой

$$f(k) = 2 \sum_{i=1}^N a_i (|i - k| + 1).$$

Раскроем знак модуля:

$$f(k) = 2 \left( \sum_{i=1}^N a_i + \sum_{i=1}^{k-1} a_i (k - i) + \sum_{i=k+1}^N a_i (i - k) \right).$$

Теперь посмотрим как изменится значение  $f$  при переходе от  $k$  к  $k + 1$ :

$$\Delta f(k) = f(k + 1) - f(k) = \sum_{i=1}^k a_i - \sum_{i=k+1}^N a_i$$

Поскольку все  $a_i \geq 0$ , функция  $\Delta f(k)$  будет возрастающей. Так как  $\Delta f(0) = -\sum a_i$ ,  $\Delta f(N) = +\sum a_i$ , то существует такое  $k^*$ , где  $\Delta f(k)$  впервые примет неотрицательное значение. Тогда ясно, что до  $k^*$  функция  $f(k)$  будет убывать, а после  $k^*$  – возрастать. А значит в точке  $k^*$  она имеет минимум. Эта точка будет медианой множества всех предметов, т.е. такой, что значение  $\sum_{i=1}^k a_i$  максимально близко к половине общего количества предметов ( $\frac{1}{2} \sum_{i=1}^k a_i$ ). Очевидно она без особого труда может быть найдена за  $O(N)$ .

Переходим теперь к двумерному случаю. Теперь функция количества ходов будет иметь вид:

$$f(k_1, k_2) = 2 \sum_{i_1, i_2=1}^{N_1, N_2} a_{i_1 i_2} (|i_1 - k_1| + |i_2 - k_2| + 1).$$

Разобьем суммируемое выражение на части и выберем для каждой из них удобный порядок суммирования:

$$f(k_1, k_2) = 2 \left( \sum_{i_1, i_2=1}^{N_1, N_2} a_{i_1 i_2} + \sum_{i_1=1}^{N_1} \left( \sum_{i_2=1}^{N_2} a_{i_1 i_2} \right) |i_1 - k_1| + \sum_{i_2=1}^{N_2} \left( \sum_{i_1=1}^{N_1} a_{i_1 i_2} \right) |i_2 - k_2| \right).$$

Первое слагаемое здесь не зависит от  $k_1$  и  $k_2$ . Второе слагаемое зависит лишь от  $k_1$ , поэтому его минимум можно находить как решение одномерной задачи для значений  $\sum_{i_2=1}^{N_2} a_{i_1 i_2}$ , то есть для суммарных количеств предметов по каждой строке. Аналогично, минимум третьего слагаемого находится как решение одномерной задачи для значений  $\sum_{i_1=1}^{N_1} a_{i_1 i_2}$ . Таким образом, достаточно найти медианы множества предметов по каждой оси. Они и будут координатами оптимального расположения склада.

## Задача I. Сила героя

Имя входного файла:	<code>i.in</code>
Имя выходного файла:	<code>i.out</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

В написанной Петей игре каждый герой имеет 4 боевых показателя: силу атаки, силу защиты, силу магии и силу знания. Изначально он имеет

некоторые показатели, которые генерируются случайно. Однако, при накоплении опыта и переходе героя на следующий уровень знаний, игроку предоставляется возможность увеличить любую из этих сил на 1.

Петя считает, что эффективность героя определяется суммой квадратов его показателей. Тогда как Вася считает, что эффективность определяется произведением этих чисел.

Коля в растерянности и просит вас помочь ему определить какой максимальной эффективности по Петиной и по Васиной формуле может добиться его герой после повышения уровня на некоторое число.

### Формат входного файла

В первой строке заданы четыре числа  $A, D, M, K$  – первоначальные боевые показатели Колиного героя. Во второй строке – количество повышений уровня. Все числа – целые неотрицательные и не превосходят 10000.

### Формат выходного файла

Выведите два целых числа – максимальная эффективность, которой может достичь герой, по Петиной формуле, и максимальная эффективность по Васиной формуле.

### Примеры

i.in	i.out
2 2 2 2 0	16 16
2 3 2 3 1	33 54
1 2 3 6 3	95 162

### Разбор задачи I. Сила героя

Формализуем постановку задачи. У нас есть 4 целых числа  $a_1, a_2, a_3, a_4$ . Без ограничения общности можно считать, что они упорядочены по неубыванию (этого всегда можно добиться за счет сортировки). Необходимо найти целые неотрицательные числа  $b_1, b_2, b_3, b_4$  такие, что  $b_1 + b_2 + b_3 + b_4 = K$  и при этом, чтобы было максимально возможным значение

$$(a_1 + b_1)^2 + (a_2 + b_2)^2 + (a_3 + b_3)^2 + (a_4 + b_4)^2.$$

Ответом для этой задачи будет  $b_1 = b_2 = b_3 = 0, b_4 = K$ . Предположим, что оптимальное значение достигается на наборе, в котором хотя бы одно

из чисел  $b_1, b_2, b_3$  отлично от нуля. Пусть это будет  $b_i$ , тогда рассмотрим как изменится величина  $(a_i + b_i)^2 + (a_4 + b_4)^2$ , если перенести  $b_i$  к  $b_4$ :

$$\begin{aligned} a_i^2 + (a_4 + b_4 + b_i)^2 &= a_i^2 + (a_4 + b_4)^2 + 2(a_4 + b_4)b_i + b_i^2 > a_i^2 + (a_4 + b_4)^2 + 2a_i b_i + b_i^2 = \\ &= (a_i + b_i)^2 + (a_4 + b_4)^2. \end{aligned}$$

Таким образом значение увеличится, а значит исходный набор  $b$  был неоптимальным.

Перейдем теперь к задача максимизации произведения  $c_1 c_2 c_3 c_4$ , где  $c_i = a_i + b_i$ . Будем считать, что  $c_1 \leq c_2 \leq c_3 \leq c_4$ . Если это не так и  $c_i > c_{i+1}$ , то можно взять новые значения  $b$  так, что эти величины поменяются местами и произведение сохранится. Такими будут значения  $b'_i = b_{i+1} + (a_{i+1} - a_i)$ ,  $b'_{i+1} = b_i - (a_{i+1} - a_i)$ . В силу того, что  $a_i + b_i = c_i > c_{i+1} \geq a_{i+1}$ , значение  $b'_{i+1}$  будет неотрицательным.

Докажем следующее утверждение. Если  $b_4 \neq 0$ , то  $c_4 - 1 \leq c_1 \leq c_4$ . Предположим противное, что при  $b_4 \neq 0$ , значение  $c_1 < c_4 - 1$ . Тогда выберем наименьшее  $i$  такое, что  $c_i = c_4$ , а  $c_{i-1} < c_4$ , и  $j$  такое, что  $c_j = c_1$ , а  $c_{j+1} > c_1$ . Ясно, что такие  $i$  и  $j$  существуют, причем  $b_i \neq 0$  (в противном случае  $c_i$  никак не могло бы достигнуть величины  $c_4$ ). Тогда возьмем новые значения  $b'_j = b_j + 1$ ,  $b'_i = b_i - 1$ . Тогда  $c'_j = c_j + 1$ ,  $c'_i = c_i - 1$  (при этом будет оставаться верным неравенство  $c'_j \leq c'_i$ ) и

$$c'_i c'_j = (c_i - 1)(c_j + 1) = c_i c_j + (c_i - c_j - 1) > c_i c_j.$$

Последнее неравенство выполнено в силу того, что  $c_i = c_4$ , а  $c_j = c_1 < c_4 - 1$ . Таким образом, изначальные значения  $b_i$  были не оптимальными. Таким образом, при  $b_4 \neq 0$  все значения  $c$  должны быть “почти” одинаковыми, т.е. отличаться друг от друга не более чем на 1. Но такое распределение строится однозначно: все  $c$  получают значения  $\lfloor (a_1 + a_2 + a_3 + a_4 + K)/4 \rfloor$ , кроме  $(a_1 + a_2 + a_3 + a_4 + K) \bmod 4$  последних, которые увеличиваются еще на 1.

Если же такое распределение невозможно, а это может быть лишь в случае, когда  $a_4$  оказывается больше, чем нужное значение  $c_4$ , следует принять  $b_4 = 0$  и рассматривать задачу для трех чисел  $a_1, a_2, a_3$ . Ее решение строится аналогичным образом – нужно обеспечить примерное равенство  $c_1, c_2, c_3$ . Если и это нельзя сделать, то принимаем  $b_3 = 0$ , и решаем задачу для двух чисел. Если же и их не удается сделать почти равными, то  $b_2 = 0$ , а  $b_1 = K$ .

## Задача J. Караваны

Имя входного файла:	j.in
Имя выходного файла:	j.out
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

“Здравствуйте. Я, Кирилл. Хотел бы чтобы вы сделали игру, ЗД-экшон суть такова... Пользователь может играть лесными эльфами, охраной дворца и злодеем. И если пользователь играет эльфами то эльфы в лесу, домики деревянные набирают солдаты дворца и злодеи. Можно грабить коровы... [skipped] P.S. Я джва года хочу такую игру.”

*Письмо в компанию по разработке игр MiST land*

Ну и конечно же Петя не мог при разработке не учесть пожеланий мирового геймерского сообщества, поэтому в его игре можно “набегать” и “грабить коровы”. Караван состоит из повозок, в каждой из которых находится определенное количество золота. Ограбление можно начать с любой повозки, продвигаясь далее к следующей или предыдущей и забирая золото, которое находится в них. Всю операцию нужно провести достаточно быстро, иначе подоспеет охрана и схватит грабителей. Поэтому забрать можно будет не из всех повозок.

Коля, играющий, как и следовало ожидать, за эльфов, знает и количество повозок в караване  $N$ , и сколько золота в каждой повозке (в  $i$ -ой повозке находится  $a_i$  золота), и максимальное количество повозок  $k$ , которое его отряд успеет обойти до прибытия охраны. Ему необходимо узнать максимальное количество золота, которое он может получить в результате операции.

### Формат входного файла

В первой строке заданы два целых числа  $N$  и  $k$  ( $1 \leq k \leq N \leq 100000$ ), определяющие размер каравана и максимальное число повозок, которые можно ограбить. Во второй строке задаются  $N$  целых чисел  $a_i$ , определяющие количество золота в повозках ( $0 \leq a_i \leq 10000$ ).

### Формат выходного файла

В единственной строке выведите наибольшую сумму, которую может принести ограбление.

## Пример

j.in	j.out
7 3 3 2 1 2 3 2 1	7

## Разбор задачи J. Караваны

В задаче требуется найти отрезок из  $K$  последовательных элементов, сумма которых будет максимально возможной. Непосредственный подсчет суммы для каждого отрезка потребует времени порядка  $O(NK)$ . Один из способов решения – реализовать дерево отрезков на сумму, тогда время сократится до  $O(N \log N)$ . Но поскольку в данной задаче не требуется запросов на изменение, то построение дерева отрезков – это неоправданные затраты. Достаточно лишь предподсчитать частичные суммы массива, т.е.

значения  $S_j = \sum_{i=1}^j a_i$ , где  $k = \overline{0, N}$ . Очевидно это можно сделать за время  $O(N)$ . Тогда сумма чисел на отрезке длины  $K$ , заканчивающемся  $j$ -ым элементом будет равна  $P_j = S_j - S_{j-K}$ . Другой способ – при проходе по массиву рекуррентно вычислять  $P_j$ : значение  $P_K$  вычисляется непосредственно, а последующие значения по формуле  $P_j = P_{j-1} + a_j - a_{j-K}$ . Остается лишь выбрать максимальное из значений  $P_j$  и выдать его в качестве результата.

## Задача K. Кто ходит в гости по утрам

Имя входного файла: k.in  
Имя выходного файла: k.out  
Ограничение по времени: 3 с  
Ограничение по памяти: 64 Мб

Кто ходит в гости по утрам,  
Тот поступает мудро.  
Известно всем, тарам-парам,  
На то оно и утро! На то оно и утро!

Скучна вечерняя пора,  
Хозяева зевают.  
Но если гость пришёл с утра,  
Такого не бывает! Такого не бывает!

Да, если гость пришёл с утра, –  
Ему спешить не надо.  
Кричат хозяева “Ура!”,  
Они ужасно рады. Они ужасно рады!

Недаром солнце в гости к нам  
Всегда приходит по утрам.  
Тарам-парам, тарам-парам,  
Ходите в гости по утрам.

Б.Заходер, “Песня Винни-Пуха”

В Чудесном Лесу живут  $N$  различных персонажей, у каждого из которых есть свой собственный домик. Следуя заветам одного из самых известных лесных персонажей, Винни-Пуха, каждый житель считает необходимым проснуться с утра пораньше, умыться, одеться и пойти в гости к кому-либо. Разумеется, чтобы поступить не просто мудро, а очень мудро и не потратить слишком много времени на дорогу, персонаж отправится не к кому-нибудь, а к своему соседу, то есть к тому из жителей, домик которого находится к данному персонажу на наименьшем возможном расстоянии. Нетрудно понять, что хозяина этого домика не окажется дома, поскольку он тоже воспользуется правилом Винни-Пуха. Лишь по этой причине некому будет ни крикнуть “Ура!”, ни обрадоваться гостям. Если вдруг окажется, что несколько домиков расположены на минимальном расстоянии от персонажа, то он выберет для похода в гости домик с наименьшим номером. Ваша задача – определить какие персонажи соберутся у каждого домика.

### **Формат входного файла**

В первой строке задается количество персонажей  $N$  ( $2 \leq N \leq 100000$ ). В каждой из последующих  $N$  строк задаются по два числа – координаты точки на плоскости, в которой расположен домик соответствующего персонажа. Все координаты – целые неотрицательные числа, не превосходящие  $10^9$ .

### **Формат выходного файла**

Выведите  $N$  строк.  $i$ -ая строка должна содержать число  $i$ , за которым следует двоеточие и далее в порядке возрастания номера персонажей, которые придут в гости в  $i$ -ый домик.

### **Пример**

<b>k.in</b>	<b>k.out</b>
6	1: 2 3
0 0	2: 1
1 0	3:
0 1	4: 5
3 3	5: 4 6
2 2	6:
3 1	

## Разбор задачи K. Кто ходит в гости по утрам

Фактически, в задаче нужно для каждого персонажа найти его ближайшего соседа. Построим для точек  $p_i = (x_i, y_i)$  диаграмму Вороного. Рассмотрим многоугольник  $V(i)$ , содержащий точку  $p_i$ . Тогда, как мы знаем из теоремы 3, каждый ближайший сосед точки  $p_i$  определяет некоторое ребро многоугольника  $V(i)$ . Таким образом, просмотрев все ребра многоугольника  $V(i)$  и проверяя расстояния до соответствующих им вершин, можно найти ближайшего соседа точки  $p_i$  с наименьшим номером. Так как каждое ребро принадлежит двум многоугольникам Вороного, то ни одно из ребер не будет просматриваться более двух раз. По следствию из теоремы 5 диаграмма Вороного имеет  $O(N)$  ребер. Таким образом, имея диаграмму Вороного, можно найти всех ближайших соседей за линейное время. На построение же самой диаграммы потребуется порядка  $O(N \log N)$  операций. Остается лишь построить для каждого  $i$  список тех точек, для которых  $p_i$  является ближайшим соседом. Очевидно, это может быть выполнено за время  $O(N)$ .

## Задача L. Принц или самозванец

Имя входного файла:	1.in
Имя выходного файла:	1.out
Ограничение по времени:	3 с
Ограничение по памяти:	64 Мб

В древние времена и минувшие века и столетия был в Персии великий царь Дарий. И процветало в то время персидское государство, и было всего в достатке. В один прекрасный день у царя родился сын, и не было в тот день на земле счастливее человека, чем Дарий. Большой праздник устроил он в честь этого события. Но пока шли все торжества по случаю рождения наследника, нанятые недругами Дария ассасины проникли в опочивальню и выкрали ребенка. Страшно разгневался Дарий и приказал казнить охрану. А кроме того, повелел царь объявить розыск и пообещал огромную награду тому, кто найдет и вернет принца во дворец...

...Шли дни, а за ними недели, месяцы и годы, но никаких вестей о сыне не поступало царю. И вот, когда пошел восемнадцатый год поисков, во дворец вошел стройный высокий юноша с искоркой во взгляде, который назывался тем самым пропавшим принцем. Невероятные истории рассказывал юноша. И о том, как он был подброшен в семью простого рыбака, которому под страхом смерти запретили говорить о принце кому бы то ни было, и о том, как он жил все эти годы, и о том, как, умирая, старый

рыбак открыл юноше страшную тайну о его происхождении.

И уже готов был Дарий поверить юноше и принять его в свои объятия, но везирь посоветовал царю не спешить, а сначала сделать проверку. Известно, что каждая клетка организма человека содержит ДНК, которая представляет из себя цепочку нуклеотидов, кодируемы символовами А, Г, Т, С, и, кроме того, что эти цепочки у близких родственников должны быть похожими. Везирь предложил взять некоторый фрагмент ДНК юноши и сравнить с ДНК царя, начиная с какой-нибудь позиции. Разумеется, наилучший вариант будет тогда, когда фрагмент в точности встретится начиная с выбранной позиции. Но, вообще говоря, мерой похожести будет считаться количество совпадений при сравнении соответствующих элементов. Везирь с царем просят вас определить позицию, с которой следует начинать сравнение для того, чтобы степень похожести была максимальной.

## **Формат входного файла**

В первой строке задается ДНК царя – последовательность символов А, Г, Т, С. Во второй строке аналогичным образом задается фрагмент ДНК юноши. Длины строк не превышают 200000 и вторая строка не длиннее первой.

## **Формат выходного файла**

Выведите номер позиции, с которой следует начинать сравнение для того, чтобы добиться максимально возможной похожести. Если таких позиций несколько, выведите первую из них. Позиции нумеруются с 1.

## **Примечание**

Заметьте, что каждому символу второй строки при сравнении должен соответствовать некоторый символ первой строки, иными словами, при размещении второй строки, начиная с нужной позиции, она не выйдет за границы первой.

## **Примеры**

<b>1.in</b>	<b>1.out</b>
AGTCAGTC GTC	2
AAGGTTCC TCAA	5

## Разбор задачи L. Принц или самозванец

Предположим сначала, что у нас есть две последовательности, состоящие из 0 и 1:  $a_0, a_1, \dots, a_{N-1}$  и  $b_0, b_1, \dots, b_{M-1}$ , где  $M \leq N$ . В качестве меры похожести  $c_j$  последовательности  $b$  на последовательность  $a$  начиная с позиции  $j$ , примем количество совпадений лишь по значению 1. Тогда

можно записать, что  $c_j = \sum_{i=0}^{M-1} a_{i+j} b_i$ . Таким образом,  $c$  – это фактически взаимнокорреляционная функция для  $a$  и  $b$ . Чтобы перейти от взаимной корреляции к свертке, развернем одну из последовательностей, то есть примем, например,  $b'_i = b_{M-1-i}$ . В таком случае  $c$  – это будет сдвинутая на  $M - 1$  свертка  $a$  и  $b'$ . Чтобы перейти к циклической свертке и воспользоваться для ее вычисления быстрым преобразованием Фурье, увеличим размеры обоих массивов до  $2N$ , заполнив новые элементы значением 0 (это обеспечит равенство линейных и циклических сверток) затем еще до ближайшей степени двойки. Тогда  $c = \text{shiftleft}(\mathcal{F}^{-1}(\mathcal{F}(a) \cdot \mathcal{F}(b)), M - 1)$ , где `shiftleft` обозначает сдвиг влево.

Перейдем теперь к решению поставленной задачи. Построим сначала последовательность  $a$  так, что  $a_i = 1$ , если соответствующий элемент ДНК царя равен  $A$ , в противном случае  $a_i = 0$ . Аналогично построим последовательность  $b$  для ДНК принца. Выполнив для них вычисление взаимнокорреляционной функции, получим количества совпадений символов  $A$  при сравнениях, начиная с каждой позиции. Аналогично найдем количества совпадений для  $G$ ,  $T$  и  $C$ . Остается лишь просуммировать эти значения для каждой позиции и выбрать наибольший результат в соответствующих пределах.

## Задача М. Что? Где? Когда?

Имя входного файла:	<code>m.in</code>
Имя выходного файла:	<code>m.out</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

“Уважаемые знатоки информатики”! В известной телевизионной игре “Что? Где? Когда?” команда знатоков играет против команды телезрителей, приславших на передачу свои вопросы. Письма с вопросами кладутся на круглый игровой стол, разделенный на  $N$  равных секторов – в каждый сектор по одному письму. В центре этого стола установлен волчок со стрелкой. Каждый раунд начинается с того, что распорядитель зала раскручивает волчок. Когда волчок останавливается, из сектора, на котором

остановилась стрелка берется конверт и ведущий зачитывает соответствующий вопрос, на который знатоки должны будут ответить после минуты обсуждения. Если же вопрос из сектора, где остановилась стрелка, уже сыграл в одном из предыдущих раундов, то выбирается следующий по часовой стрелке еще не игравший вопрос. Вообще говоря, в телепередаче игра идет до тех пор пока одна из команд не наберет определенного количества очков, но мы будем считать, что игра заканчивается только тогда, когда на столе не останется ни одного вопроса.

Допустим уже прошло несколько раундов и вопросы из некоторых секторов уже сыграли. “А теперь внимание вопрос”! (удар гонга!)

За одну секунду ваша программа должна ответить, какая вероятность того, что в  $k$ -ом (начиная с текущего) раунде будет играть вопрос, находящийся в  $i$ -ом секторе. Разумеется, поскольку секторы одинаковы, то и остановку стрелки волчка в каждом из них считаем равновероятной.

## Формат входного файла

В первой строке задаются три целых числа  $N$ ,  $i$ ,  $k$  ( $1 \leq i \leq N \leq 20$ ,  $1 \leq k \leq N$ ). Во второй строке задаются  $N$  чисел, каждое из которых равно либо 0, либо 1. Значение 0 обозначает, что вопрос из соответствующего сектора уже сыграл в одном из предыдущих раундов, 1 – вопрос пока еще на столе.

## Формат выходного файла

Выведите вероятность того, что вопрос из  $i$ -ого сектора сыграет после  $k$ -го вращения волчка с точностью не менее  $10^{-8}$ .

## Примеры

<b>m.in</b>	<b>m.out</b>
10 5 1	0.1
1 1 1 1 1 1 1 1 1	
4 2 2	0.25
0 1 1 0	

## Разбор задачи М. Что? Где? Когда?

Нетрудно догадаться, что это задача на динамическое программирование по профилю. Мы можем всегда свести задачу к  $i = 1$  (т.е. к вычислению вероятности для первого сектора), для этого необходимо выполнить циклический сдвиг исходной ситуации с вопросами на столе. Пусть

$P(k, m)$  – вероятность того, что первый вопрос выпадет после  $k$ -го вращения волчка, если исходная ситуация описывается массивом  $m$  (разумеется, при реализации  $m$  будет битовой маской). Считаем, всегда, что  $m_1 = 1$ . Если это не так, вероятность будет равна 0 – вопрос уже выпал раньше. Тогда начальными значениями этой функции можно выбрать  $P(1, m) = (c_1 + 1)/N$ , где  $c_1$  – количество сыгравших секторов перед первым или, что то же самое, длина непрерывной последовательности в конце массива  $m$ . Построим теперь рекуррентную формулу для значений  $k > 1$ . Вероятность того, что выпадет вопрос с номером  $j$  (если он еще не выпадал, то есть  $m_j = 1$ ) равна  $(c_j + 1)/N$ , где  $c_j$  – длина непрерывной последовательности нулей перед  $j$ -ой позицией. Если этот вопрос сыграет, то вероятность выпадения вопроса 1 еще через  $k - 1$  раунд будет равна  $P(k - 1, \text{reset}(m, j))$ , где  $\text{reset}(m, j)$  – это массив, все элементы которого совпадают с соответствующими элементами  $m$ , кроме  $j$ -ого, который принимается равным 0. Тогда по формуле полной вероятности получим:

$$P(k, m) = \frac{1}{N} \sum_{j: m_j=1} (c_j + 1) P(k - 1, \text{reset}(m, j)).$$

В задаче есть лишь один подводный камень – номер раунда может оказаться больше чем количество оставшихся вопросов на столе. В этом случае, искомая вероятность очевидно равна нулю. Всего потребуется вычислить не более, чем  $2^N$  вероятностей, каждая из которых вычисляется за время  $O(N)$ . Поэтому общая сложность алгоритма будет  $O(N2^N)$ .

### Задача N. Укладка плит

Имя входного файла:	n.in
Имя выходного файла:	n.out
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Известный программист Петя вновь принялся за написание новой компьютерной игры в стиле платформер. На одном из уровней у него есть коридор, разбитый на  $N$  равных участков. Этот коридор должен быть покрыт плитами. Одна плита может иметь любую длину и соответственно покрывать несколько последовательных участков. Требуется выполнить укладку плит таким образом, чтобы каждый участок был покрыт заданным числом плит. Помогите Пете сосчитать, какое минимальное количество плит, которое ему понадобится для этого.

## Формат входного файла

В первой строке задается целое число  $N$  ( $1 \leq N \leq 200000$ ) – длина коридора. Во второй строке записано  $N$  целых чисел, каждое из которых определяет количество плит, которыми должен покрываться соответствующий участок. Все числа неотрицательные и не превышают  $10^9$ .

## Формат выходного файла

Выполните минимальное количество плит, которое понадобится для укладки.

## Примеры

<b>n.in</b>	<b>n.out</b>
3 3 4 1	4
3 4 1 3	6

## Разбор задачи N. Укладка плит

Наиболее очевидное решение заключается в следующем. Найдем минимальное значение в заданном массиве – пусть это будет  $a_i$ . Ясно, что только  $a_i$  плит мы можем положить на целый коридор. И тогда нам останется выложить рекурсивно участки с 1 по  $i - 1$ , а также участки с  $i + 1$  по  $N$ . Таким образом, алгоритм будет очень похожим на алгоритм быстрой сортировки. Но поскольку здесь в разбиении на части нет никакого произвола, без труда строятся примеры, в которых непосредственное нахождение минимума приведет к общей сложности алгоритма порядка  $O(N^2)$ . Тем не менее, можно ускорить нахождение минимума, построив на массиве  $a$  дерево отрезков на минимум, что позволит сократить количество операций до  $O(N \log N)$ . Однако есть и более простое решение.

Пусть  $f(i)$  – количество плит, которые необходимы для укладки участков с 1 по  $i$ . Тогда наша задача заключается в нахождении  $f(N)$ . Начальное значение  $f(1) = a_1$ . Посмотрим как можно выразить значение  $f(i)$  через предыдущее. Если  $a_i \leq a_{i-1}$ , то  $a_i$  из плит, которыми укладывался участок  $i - 1$  могут быть использованы и для покрытия  $i$ -го участка. Если же  $a_i > a_{i-1}$ , то максимум  $a_{i-1}$  плит мы можем продлить на  $i$ -ый участок, а еще  $a_i - a_{i-1}$  плит придется уложить новых. Таким образом,

$$f(i) = \begin{cases} f(i-1) & \text{при } a_i \leq a_{i-1}, \\ f(i-1) + a_i - a_{i-1} & \text{при } a_i > a_{i-1}. \end{cases}$$

Сложность такого алгоритма будет  $O(N)$ . Его форму можно упростить. Если положить  $a_0 = 0$ , то ответом к задаче будет фактически сумма всех положительных разностей  $a_i - a_{i-1}$ .

## Задача О. Прохождение коридора

Имя входного файла: `o.in`

Имя выходного файла: `o.out`

Ограничение по времени: 1 с

Ограничение по памяти: 64 Мб

Как мы уже знаем, в игре Пети есть коридор, разбитый на  $N$  участков. Предположим, что каждый из участков покрыт некоторым числом единичных плит. Персонаж в игре, которым управляет игрок, находится в начале коридора перед первым участком и может пройти по этому участку, потратив на это один ход. Если на нем была хотя бы одна плита, то после прохождения по участку одна плита с него исчезает. Таким образом количество плит уменьшится на 1. Если же на участке не было ни одной плиты, то персонаж погибает, соответственно игрок теряет одну жизнь, после чего на этом участке появляется  $K$  новых плит, а у игрока появляется новый персонаж в начале коридора. Если игрок удачно прошел участок и не погиб, то он оказывается перед следующим участком, который он может пройти, если на нем есть хотя бы одна плита, или погибнуть, если плит нет. В любом случае потребуется один ход. Разрешается лишь движение вперед. Считается, что игрок прошел коридор, если его персонаж в какой-то момент окажется в конце коридора, то есть пройдет последний участок и не погибнет на нем. Помогите игроку узнать, сколько потребуется жизней и ходов для прохождения коридора.

### Формат входного файла

В первой строке даны два целых числа  $N$  и  $K$  ( $1 \leq N \leq 10000$ ,  $1 \leq K \leq 100$ ) – длина коридора и количество появляющихся после гибели персонажа плит на участке. Во второй строке записано  $N$  целых чисел, каждое из которых определяет количество плит, которыми покрыт изначально соответствующий участок. Эти числа могут принимать значения от 0 до  $K$  включительно.

### Формат выходного файла

Выполните сколько жизней потеряет игрок и сколько ходов он сделает до того момента, когда его персонаж попадет в конец коридора.

## Примеры

<b>o.in</b>	<b>o.out</b>
3 3	0 3
2 2 2	
4 2	2 7
1 0 2 1	
5 1	31 62
0 0 0 0 0	

## Разбор задачи О. Прохождение коридора

Обозначим  $f(i)$  – количество раз, которое персонаж игрока будет проходить по участку с номером  $i$ . Тогда ясно, что количество ходов будет равно сумме всех  $f(i)$ , а количество потраченных жизней  $f(1) - 1$ . Попытка находить  $f(i)$  в прямом порядке приводит к большому количеству модулярных неравенств, которые, по-видимому, будет совсем непросто решить. Попробуем находить их с конца. Для упрощения записи введем еще фиктивный  $(N + 1)$ -ый участок, который, фактически, и будет концом коридора. Как только персонаж игрока попадает на этот участок, он прошел весь коридор. Поэтому  $f(N + 1) = 1$ . Пусть мы знаем сколько раз персонаж должен проходить  $(i + 1)$ -ый участок. Найдем такое число  $T$ , что при  $f(i) = A$  персонаж  $f(i + 1)$  раз пройдет участок  $i$  без потерь жизней. Это означает, что в последовательности  $a_i, a_i - 1, \dots, a_i - A + 1$  должно быть  $f(i + 1)$  чисел, некратных  $K + 1$ . Нетрудно найти, что кратных будет  $\lceil (A - a_i)/(K + 1) \rceil$ . А это значит, что

$$A - \left\lceil \frac{A - a_i}{K + 1} \right\rceil = f(i + 1).$$

Отсюда

$$A - \frac{A - a_i}{K + 1} \geq f(i + 1),$$

и

$$AK \geq f(i + 1)(K + 1) - a_i.$$

Наименьшее значение  $A$ , которое удовлетворяет этому неравенству, и будет искомым значением  $f(i)$ :

$$f(i) = \left\lceil \frac{f(i + 1)(K + 1) - a_i}{K} \right\rceil.$$

## Задача Р. Игра

Имя входного файла: **p.in**  
Имя выходного файла: **p.out**  
Ограничение по времени: 1 с  
Ограничение по памяти: 64 Мб

Двое игроков играют в следующую игру. На столе лежит  $N$  кучек камней, в  $i$ -той кучке в начале  $n_i$  камней, кроме того ей приписаны натуральные числа  $x_i$  и  $y_i$ . Игроки ходят по очереди. За один ход игрок выбирает какую-то кучку. Пусть ее номер  $i$ . Тогда он может взять из нее либо  $x_i$ , либо  $y_i$  камней. Ход может быть выполнен, если кучка содержит не меньше камней, чем игрок собирается из нее взять. Проигрывает тот, кто не может сделать ход. Определите, кто выиграет при правильной игре: игрок, который ходит первым, или игрок, который ходит вторым.

### Формат входного файла

В первой строке входного файла задано натуральное число  $N \leq 10000$ . Следующие  $N$  строк содержат по 3 числа каждая. А именно,  $i$ -я строка содержит параметры  $i$ -той кучки:  $n_i$ ,  $x_i$ ,  $y_i$ . При этом  $1 \leq n_i, x_i, y_i \leq 10^{18}$ .

### Формат выходного файла

В единственную строку выходного файла выведите “First” (без кавычек), если выигрывает первый игрок, и “Second” (без кавычек) иначе.

### Примеры

<b>p.in</b>	<b>p.out</b>
4 3 1 2 1 2 3 4 1 1 10 3 5	Second

## Разбор задачи Р. Игра

Посчитаем для  $i$ -той кучки числа Шпрага-Гранди. Напомним их определение. В данном случае число Шпрага-Гранди  $G_i(n)$   $i$ -той кучки, если в ней  $n$  камней, равно минимальному неотрицательному числу, которое не равно числу  $G_i(n - x_i)$ , если  $x_i \leq n$ , и не равно числу  $G_i(n - y_i)$ , если  $y_i \leq n$ . По теореме Шпрага-Гранди первый игрок выиграет тогда и только тогда, когда число  $G_1(n_1) \text{ xor } G_2(n_2) \text{ xor } \dots \text{ xor } G_N(n_N)$  не равно нулю. Здесь  $a \text{ xor } b =$

результат побитового сложения чисел  $a$  и  $b$  по модулю 2. Таким образом, задача сводится к эффективному нахождению чисел  $G_i(n_i)$ .

Пусть  $x, y$  фиксированные натуральные числа, причем  $x \leq y$ . Мы рассматриваем следующую последовательность:

$$G(n) = -1 \quad \text{при } n < 0$$

и

$$G(n) = \min\{\{0, 1, 2, \dots\} \setminus \{G(n-x), G(n-y)\}\} \quad \text{при } n \geq 0.$$

Ясно, что  $G(n) = [n/x] \bmod 2$  при  $n < y$ . Пусть  $z$  кратно  $x$  и удовлетворяет неравенствам  $y < z \leq x+y$ . Очевидно, что такое  $z$  существует и единственное. Ясно, что при  $y \leq n < x+y$  имеем  $G(n) \neq 0$ , так как  $G(n-y) = 0$  для всех таких  $n$ . Ясно, что  $G(n) \neq G(n-x)$ . При  $y \leq n < z$  имеем  $G(n-x) = G(y-x) = [(y-x)/x] \bmod 2 = 1 - [y/x] \bmod 2$ . При  $z \leq n < y+x$  имеем  $G(n-x) = G(z-x) = [(z-x)/x] \bmod 2 = 1 - [z/x] \bmod 2$ . Так как  $y < z \leq x+y$  и  $z$  кратно  $x$ , то  $[y/x]$  и  $[z/x]$  разной четности. Тем самым мы можем за  $O(1)$  находить  $G(n)$  при  $n < x+y$ . Кроме того, если аккуратно рассмотреть случаи, то можно заметить, что  $G(n) = [n/x] \bmod 2$  при всех  $n$  от 0 до  $x+y-1$ , кроме случая когда  $n \geq y$  и  $G(n) = 0$ . Для таких  $n$  надо заменить значение  $G(n)$  на 2.

Теперь докажем по индукции, что последовательность  $G(n)$  периодическая с периодом  $x+y$ . Если мы рассматриваем какой-то отрезок от  $q(x+y)$  до  $q(x+y) + x + y - 1$  и уже доказали утверждение для чисел от  $q(x+y)$  до  $q(x+y) + y - 1$ , то при  $q(x+y) + y \leq n < q(x+y) + x + y$  доказательство будет таким же как для диапазона  $y \leq n < x+y$ , так как никакой ход для таких  $n$  не затрагивает числа меньшие  $q(x+y)$ . С числами от  $q(x+y)$  до  $q(x+y) + y - 1$  все несколько сложнее, но если аккуратно разобрать все случаи, то мы получим требуемый результат.

Таким образом, мы умеем легко за  $O(1)$  находить  $G_i(n_i)$ . Поэтому задача решена.

### Задача Q. Гамильтонов цикл

Имя входного файла:	<code>q.in</code>
Имя выходного файла:	<code>q.out</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Однажды граф Уильям Гамильтон решил совершить кругосветное путешествие, в котором он посетил бы  $N$  крупнейших городов Земли. Между

всеми парами городов есть дороги. Каждой дороге, ведущей от одного города до другого, граф Гамильтон приписывает некоторое число (зрелищность), которое является степенью числа 2. Поскольку по разные стороны от одной дороги пейзажи различны, то зрелищность дороги при проезде по ней в одну сторону отличается от зрелищности при проезде в другую сторону. Более того, оказалось, что все дороги имеют различную зрелищность. Граф хочет выбрать замкнутый маршрут (начинающийся и заканчивающийся в одном и том же городе), проходящий через все города по одному разу и имеющий максимальную суммарную зрелищность. Найдите оптимальный замкнутый маршрут для графа Гамильтона. Граф живет в городе с номером 1.

### **Формат входного файла**

В первой строке задается целое число  $N$  — количество городов ( $2 \leq N \leq 500$ ). В каждой из последующих  $N$  строк задается по  $N$  чисел. Если  $j$ -ое число в  $i$ -ой строке равно  $c_{ij}$ , то зрелищность дороги из города  $i$  в город  $j$  равна  $2^{c_{ij}}$ . Все числа  $c_{ij}$  ( $i \neq j$ ) различны и находятся в диапазоне от 0 до  $10^6$  включительно. Значения  $c_{ii}$  всегда задаются равными  $-1$ , что означает, что дороги из города  $i$  в него же, не проходящей через какой-либо другой город, не существует.

### **Формат выходного файла**

Выведите номера всех городов в той последовательности, в которой графу Гамильтону следует их посетить, чтобы его маршрут имел наибольшую зрелищность. Граф должен начать свое путешествие в городе 1 и закончить в нем же. В случае, если существует несколько оптимальных маршрутов, можно вывести любой из них.

### **Примеры**

<b>q.in</b>	<b>q.out</b>
3 -1 5 1 4 -1 2 6 0 -1	1 2 3 1

### **Разбор задачи Q. Гамильтонов цикл**

Ясно, что зрелищность любой дороги на Земле больше, чем сумма зрелищностей всех дорог с меньшей зрелищностью. Поэтому самый зрелищный маршрут получается жадной стратегией: на каждом шаге выбирать

самую зрелищную дорогу, добавление которой в текущий набор дорог позволяет его каким-либо способом достроить до замкнутого кругосветного маршрута. Действительно, если мы не добавим эту дорогу, то любой маршрут без нее будет менее зрелищным по замечанию выше. Так как граф полный, то любое объединение непересекающихся цепочек можно достроить до замкнутого маршрута по всем городам.

Таким образом, мы получаем следующий алгоритм нахождения гамильтонова цикла. Будем просматривать все ребра в порядке убывания их зрелищности. Наш текущий недостроенный маршрут будет поддерживаться в виде объединения непересекающихся цепочек. Для каждого города  $u$  будем поддерживать величины  $\text{next}[u]$  и  $\text{prev}[u]$ , которые указывают на то, какой город следующий и предыдущий в цепочке, где лежит  $u$ . В начале эти величины равны нулю для всех городов. Тогда очередное ребро  $(u, v)$  можно достроить, если  $\text{next}[u] = 0$  и  $\text{prev}[v] = 0$ , при условии, что это ребро не соединяет конец и начало какой-то одной цепочки, кроме случая когда это последнее ребро маршрута. Поэтому надо еще поддерживать счетчик добавленных ребер и величины  $\text{first}[u]$  и  $\text{last}[u]$ , указывающие на начало и конец цепочки, где находится город  $u$ . В начале  $\text{first}[u] = \text{last}[u] = u$  для всех городов. Главное чтобы  $\text{first}[u]$  было корректно определено для любого конца цепочки, а  $\text{last}[u]$  – для любого начала. Тогда при добавлении очередного ребра нужно еще проверять дополнительно, что  $\text{first}[u] \neq v$ , кроме случая добавления последнего ребра в маршрут. И если ребро будет добавлено, то надо сделать присвоения  $\text{next}[u] := v$ ,  $\text{prev}[v] := u$ ,  $\text{first}[\text{last}[v]] := \text{first}[u]$ ,  $\text{last}[\text{first}[u]] := \text{last}[v]$ . В итоге получаем алгоритм со сложностью либо  $O(n^2 \log n)$ , если ребра сортировать быстрой, пирамидальной сортировкой или сортировкой слиянием, либо со сложностью  $O(\max C + n^2)$ , где  $\max C$  максимальная зрелищность дороги, если сортировать сортировкой подсчетом. Оба эти способа укладываются по времени.

## Задача R. Очень дружная группа

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

В классе  $N$  девочек и  $M$  мальчиков. Классному руководителю Снежане Денисовне надо выбрать группу из  $L$  девочек и  $K$  мальчиков на представление класса. Эта группа должна быть очень дружной, то есть каждый выбранный мальчик должен дружить с каждой выбранной девочкой. Снежану Денисовну интересуют все способы это сделать, а потом она выберет

из них наилучший с ее точки зрения. Помогите ей и найдите общее число способов сформировать очень дружную группу детей. Так как ответ может быть очень большим выведите его по модулю 1000000007.

### Формат входного файла

В первой строке входного файла заданы натуральные числа  $N, M, L, K$ . При этом  $L \leq N \leq 100000$  и  $K \leq M \leq 15$ . Следующие  $N$  строк содержат по  $M$  символов 0 или 1 каждая. При этом  $j$ -й символ  $i$ -й строки равен 1 тогда только тогда когда  $i$ -я девочка дружит с  $j$ -м мальчиком.

### Формат выходного файла

В единственную строку выходного файла выведите ответ на задачу.

### Примеры

<b>r.in</b>	<b>r.out</b>
4 3 1 1 111 101 110 010	8
4 3 2 1 111 101 110 010	7

### Разбор задачи R. Очень дружная группа

Зафиксируем каких-то  $K$  мальчиков из  $M$  имеющихся. Тогда девочки, которые подходят для дополнения выбранных мальчиков до очень дружной группы, должны дружить с каждым из них. Количество таких девочек легко найти:  $i$ -тая девочка подходит, если в матрице дружбы во всех столбцах соответствующих выбранным мальчикам в  $i$ -той строке находятся единицы. Если обозначить это количество через  $D$ , то количество очень дружных групп с данным набором мальчиков будет равно  $\binom{D}{L}$ , где  $\binom{n}{k} = n!/(k!(n-k)!)$  – число сочетаний из  $n$  по  $k$ . Таким образом, ответ равен сумме  $\binom{D}{L}$  по всем наборам из  $K$  мальчиков, где  $D$  – количество подходящих девочек для данного набора мальчиков.

Чтобы эффективно считать  $\binom{D}{L}$  необходимо сохранить в массиве значения  $\binom{D}{L}$  при  $0 \leq D \leq N$  до рассмотрения подмножеств мальчиков. Это

можно сделать по формулам:  $\binom{D}{L} = 0$  при  $0 \leq D < L$ ,  $\binom{D}{L} = 1$  при  $D = L$  и  $\binom{D}{L} = \binom{D-1}{L} D / (D - L)$  при  $L < D \leq N$ . Так как вычисления надо проводить по простому модулю  $p = 1000000007$ , то в последней формуле надо проводить деление по модулю  $p$ , то есть домножать на  $\text{inv}_p(D - L)$ , где  $\text{inv}_p(x)$  определяется условием  $x \cdot \text{inv}_p(x) \equiv 1 \pmod{p}$  и может быть найдено с помощью алгоритма быстрого возведения в степень по формуле  $\text{inv}_p(x) = x^{p-2} \pmod{p}$ , либо с помощью расширенного алгоритма Евклида. Оба эти способа имеют сложность  $O(\log p)$ .

Таким образом, пока мы получаем алгоритм со сложностью  $O(NM + N \log p + \binom{M}{K} KN)$ . Здесь  $O(NM)$  операций уходит на считывание данных,  $O(N \log p)$  операций на нахождение значений  $\binom{D}{L}$  при  $0 \leq D \leq N$ . На последнем этапе мы перебираем все  $\binom{M}{K}$  подмножеств мощности  $K$  множества мальчиков класса и для каждого такого подмножества находим число  $D$ , просматривая  $K$  соответствующих столбцов таблицы дружбы, и прибавляем к ответу  $\binom{D}{L}$ , беря это значение из предварительно посчитанного массива. Но такой алгоритм не укладывается по времени.

Будем хранить таблицу дружбы следующим образом. В  $j$ -том столбце заменим значения  $a_{32k+1,j}, a_{32k+2,j}, \dots, a_{32k+32,j}$  на одно число 32-битового беззнакового типа  $b_{k,j} = a_{32k+1,j} + a_{32k+2,j}2 + \dots + a_{32k+32,j}2^{31}$  для всех допустимых  $k$  (для последнего  $k$  сумма может содержать меньше 32-х слагаемых). Тогда поиск числа  $D$  для данного набора мальчиков может быть значительно ускорен. А именно, для данного набора столбцов  $1 \leq j_1 \leq j_2 \leq \dots \leq j_K \leq M$  посчитаем числа  $c_k = b_{k,j_1} \text{ and } \dots \text{ and } b_{k,j_K}$ , где  $x \text{ and } y$  – это побитовое “и” чисел  $x$  и  $y$ . Тогда  $D = \sum_k \text{bitcnt}(c_k)$ , где  $\text{bitcnt}(x)$  – число единичных битов в двоичной записи числа  $x$ . Если искать  $\text{bitcnt}(c_k)$  наивно, то пока получается алгоритм со сложностью  $O\left(NM + N \log p + \binom{M}{K}(KN/32 + N)\right)$ . Вполне возможно, что такой алгоритм, должным образом оптимизированный, может пройти по времени. Но его можно еще значительно ускорить. Сохраним значения  $\text{bitcnt}(x)$  при  $0 \leq x < 2^{16}$  в массиве. Ясно, что при  $0 \leq x < 2^{32}$  выполнено равенство  $\text{bitcnt}(x) = \text{bitcnt}(x \bmod 2^{16}) + \text{bitcnt}([x/2^{16}])$ , при этом числа  $x \bmod 2^{16}$  и  $[x/2^{16}]$  лежат в диапазоне  $[0, 2^{16})$ . Это позволит нам вычислять значения  $\text{bitcnt}(c_k)$  за  $O(1)$ . В итоге мы получим алгоритм со сложностью  $O\left(2^{16} + NM + N \log p + \binom{M}{K} KN/32\right)$ , что вполне приемлемо при данных ограничениях на  $N, M, L, K$ .

## Задача S. Деревья с нечетным числом независимых множеств

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

Вам дано натуральное число  $n \leq 1000$  и простое число  $p$  ( $10^7 < p < 10^9$ ). Найдите количество корневых деревьев из  $n$  вершин с непомеченными вершинами, имеющих нечетное число независимых множеств. Результат выведите по модулю  $p$ .

Дерево называется **корневым с непомеченными вершинами**, если какая-то его вершина зафиксирована как корень, а порядок сыновей для любой вершины не важен. То есть два дерева считаются равными, если они совпадают после некоторого переупорядочения некорневых вершин. Совсем формальное определение: два корневых дерева с непомеченными вершинами  $T_1$  и  $T_2$  равны, если существует взаимно однозначное отображение  $f$  из множества вершин дерева  $T_1$  на множество вершин дерева  $T_2$ , которое переводит корень  $T_1$  в корень  $T_2$  и для любого ребра  $(u, v)$  из  $T_1$  в  $T_2$  есть ребро  $(f(u), f(v))$ .

Некоторое множество вершин графа (возможно пустое) называется **независимым**, если никакие две вершины этого множества не соединены ребром.

### Формат входного файла

В единственной строке входного файла заданы числа  $n$  и  $p$ .

### Формат выходного файла

В единственную строку выходного файла выведите ответ на задачу.

### Примеры

<b>s.in</b>	<b>s.out</b>
1 176531359	0
2 896663687	1
3 793877167	2
51 120107707	114046817

## Разбор задачи S. Деревья с нечетным числом независимых множеств

Научимся сначала считать число независимых множеств для фиксированного дерева. Это довольно нетрудное упражнение на динамическое программирование. Будем называть независимое множество некорневым, если корень дерева не входит в него. Для поддерева с корнем в вершине  $u$  пусть  $\text{total}(u)$  — это общее число независимых множеств, а  $\text{nonroot}(u)$  — это общее число некорневых независимых множеств поддерева. Тогда нетрудно понять, что

$$\text{nonroot}(u) = \prod_{v \in \text{sons}(u)} \text{total}(v), \quad (1)$$

$$\text{total}(u) = \prod_{v \in \text{sons}(u)} \text{total}(v) + \prod_{v \in \text{sons}(u)} \text{nonroot}(v), \quad (2)$$

где  $\text{sons}(u)$  — множество сыновей вершины  $u$ .

Введем следующие три класса корневых деревьев с непомеченными вершинами:

- класс  $T_0$  — множество деревьев, у которых нечетное число независимых множеств и нечетное число некорневых независимых множеств.
- класс  $T_1$  — множество деревьев, у которых нечетное число независимых множеств.
- класс  $T_2$  — множество деревьев, у которых нечетное число некорневых независимых множеств.

Под  $T_i(n)$  будем понимать множество всех деревьев класса  $T_i$ , имеющих  $n$  вершин.

Неупорядоченный набор корневых деревьев с непомеченными вершинами класса  $T_i$  будем называть лесом класса  $i$ . Обозначим через  $F_i(n, k)$  множество лесов класса  $i$  таких, что общее число вершин в нем равно  $n$ , а максимальное число вершин в каким-либо дереве леса не превосходит  $k$ .

Из формулы (1) следует, что дерево принадлежит классу  $T_2$  тогда и только тогда, когда поддеревья этого дерева, корнями которых являются сыновья корня, принадлежат классу  $T_1$ . Поэтому если мы удалим из дерева класса  $T_2(n+1)$  его корень с выходящими из него ребрами, то мы получим лес из множества  $F_1(n, n)$  и наоборот. То есть

$$|T_2(n+1)| = |F_1(n, n)|. \quad (3)$$

Из формул (1) и (2) следует, что дерево принадлежит классу  $T_0$  тогда и только тогда, когда поддеревья этого дерева, корнями которых являются сыновья корня, принадлежат классу  $T_1$ , и хотя бы одно из них не принадлежит классу  $T_0$ . Поэтому если мы удалим из дерева класса  $T_1(n+1)$  его корень с выходящими из него ребрами, то мы получим лес из множества  $F_1(n, n) \setminus F_0(n, n)$  и наоборот. То есть

$$|T_0(n+1)| = |F_1(n, n)| - |F_0(n, n)|. \quad (4)$$

Аналогичными рассуждениями можно получить, что

$$|T_1(n+1)| = (|F_1(n, n)| - |F_0(n, n)|) + (|F_2(n, n)| - |F_0(n, n)|). \quad (5)$$

Таким образом, если мы научимся находить  $|F_i(n, k)|$ , то задача будет решена. Это нетрудная задача. Пусть в нашем наборе деревьев ровно  $j$  имеют размер  $k$ . Их можно выбрать  $\binom{|T_i(k)|+j-1}{j}$  способами (число сочетаний с повторениями), а когда мы их удалим, то получим произвольный лес из множества  $F_i(n-jk, k-1)$ . Поэтому

$$|F_i(n, k)| = \sum_{j=0}^{\lfloor n/k \rfloor} \binom{|T_i(k)| + j - 1}{j} |F_i(n - jk, k - 1)|. \quad (6)$$

Все полученные формулы позволяют находить ответ, то есть величину  $|T_1(n)|$ , за время  $O(n^2 \log n)$ . Для этого следует хранить также значения  $C_i(k, j) = \binom{|T_i(k)|+j-1}{j}$  в массиве, так как пересчет их по ходу вычисления формулы (6) увеличит сложность до  $O(n^2 \log^2 n)$ . Подробности работы с числами сочетанийсмотрите в разборе задачи “Очень дружная группа”.

## Задача Т. Максимальная степень простого

Имя входного файла: `t.in`

Имя выходного файла: `t.out`

Ограничение по времени: 3 с

Ограничение по памяти: 64 Мб

Вам дано натуральное число  $n > 1$ . Рассмотрим все различные простые делители  $n$ . Каждый из них входит в разложение  $n$  на простые множители в какой-то степени. Требуется найти среди показателей этих степеней максимальный.

### Формат входного файла

В первой строке входного файла задано натуральное число  $T \leq 500$ , количество натуральных чисел  $n$  в файле. В последующих  $T$  строках заданы сами эти числа. Гарантируется, что каждое из них не превосходит  $10^{18}$ .

## Формат выходного файла

Для каждого натурального числа  $n$  из входного файла выведите в отдельной строке максимальную степень вхождения простого числа в разложение  $n$  на простые множители.

## Примеры

<b>t.in</b>	<b>t.out</b>
5	1
2	2
12	3
108	2
36	16
65536	

## Разбор задачи Т. Максимальная степень простого

Обозначим искомый максимальный показатель для числа  $n$  через  $d_{max}(n)$ . Если натуральное число является произведением не более двух простых чисел, то оно имеет вид либо  $p$ , либо  $p^2$ , либо  $pq$ , где  $p$  и  $q$  различные простые числа. Отсюда легко видеть, что в этом случае  $d_{max}(n) = 2$  если  $n$  является точным квадратом и  $d_{max}(n) = 1$  иначе. Используя это наблюдения, можно уже получить эффективный алгоритм для вычисления  $d_{max}(n)$  для любого  $n > 1$ .

Будем просматривать все простые числа подряд. Пусть  $p$  – это очередное простое число. Если  $p^3 > n$  то выходим из цикла, иначе если  $p$  делит текущее значение  $n$ , то делим  $n$  на  $p$  пока можем, считая сколько раз мы поделили, и обновляем ответ. Легко видеть, что когда мы выйдем из цикла, текущее значение  $n$  будет произведением не более двух простых чисел (иначе для меньшего из них выполнялось бы неравенство  $p^3 \leq n$ ). Если  $n = 1$ , то ответ уже найден. Иначе мы просто проверяем, является ли  $n$  квадратом, и в этом случае обновляем ответ числом 2, и иначе числом 1.

Нам нужны простые числа до  $\sqrt[3]{n} \leq 10^6$ . Их легко можно найти и запомнить до обработывания тестов с помощью решета Эратосфена. Всего их будет немного меньше 80000. В итоге сложность алгоритма будет равна  $O(\sqrt[3]{n} \log \log n + T\pi(\sqrt[3]{n}))$ , где  $\pi(x)$  – количество простых чисел не больших  $x$ . Здесь первое слагаемое соответствует решету Эратосфена. По замечанию выше  $T\pi(\sqrt[3]{n}) < 500 \cdot 80000 = 40 \cdot 10^6$ . То есть алгоритм с такой сложностью вполне уложится по времени.

## Задача U. Минимальная степень простого

Имя входного файла: **u.in**  
Имя выходного файла: **u.out**  
Ограничение по времени: 5 с  
Ограничение по памяти: 64 Мб

Вам дано натуральное число  $n > 1$ . Рассмотрим все различные простые делители  $n$ . Каждый из них входит в разложение  $n$  на простые множители в какой-то степени. Требуется найти среди показателей этих степеней минимальный.

### Формат входного файла

В первой строке входного файла задано натуральное число  $T \leq 100000$ , количество натуральных чисел  $n$  в файле. В последующих  $T$  строках заданы сами эти числа. Гарантируется, что каждое из них не превосходит  $10^{18}$ .

### Формат выходного файла

Для каждого натурального числа  $n$  из входного файла выведите в отдельной строке минимальную степень вхождения простого числа в разложение  $n$  на простые множители.

### Примеры

<b>u.in</b>	<b>u.out</b>
5	1
2	1
12	2
108	2
36	16
65536	

## Разбор задачи U. Минимальная степень простого

Обозначим искомый минимальный показатель для числа  $n$  через  $d_{min}(n)$ . Если натуральное число является произведением не более четырёх простых чисел, то оно имеет один из следующих 10-ти видов:  $p$ ,  $p^2$ ,  $pq$ ,  $p^3$ ,  $p^2q$ ,  $pqr$ ,  $p^4$ ,  $p^3q$ ,  $p^2q^2$ ,  $p^2qr$ ,  $pqrs$ , где  $p, q, r, s$  различные простые числа. Легко видеть, что если число  $n$  имеет один из этих 10-ти видов и  $d_{min}(n) > 1$ , то оно обязательно является степенью натурального числа с показателем большим единицы. Причем если  $n$  является 4-й степенью, то

$d_{min}(n) = 4$ , если 3-й, то  $d_{min}(n) = 3$ , если  $n$  является квадратом, но не является 4-й степенью, то  $d_{min}(n) = 2$ . Используя это наблюдения, можно уже получить эффективный алгоритм для вычисления  $d_{min}(n)$  для любого  $n > 1$ .

Будем просматривать все простые числа подряд. Пусть  $p$  – это очередное простое число. Если  $p^5 > n$ , то выходим из цикла, иначе если  $p$  делит текущее значение  $n$ , то делим  $n$  на  $p$  пока можем, считая сколько раз мы поделили, и обновляем ответ. Легко видеть, что когда мы выйдем из цикла, текущее значение  $n$  будет произведением не более четырёх простых чисел (иначе для меньшего из них выполнялось бы неравенство  $p^5 \leq n$ ). Если  $n = 1$ , то ответ уже найден. Иначе мы последовательно проверяем, является ли  $n$  4-й, 3-й или второй степенью, и обновляем ответ соответствующим образом согласно замечанию выше.

Нам нужны простые числа до  $\sqrt[5]{n} \leq 10^{18/5} < 4000$ . Их легко можно найти и запомнить до обрабатывания тестов с помощью решета Эратосфена. Всего их будет немного меньше 550. В итоге сложность алгоритма будет равна  $O(T\pi(\sqrt[5]{n}))$ , где  $\pi(x)$  – количество простых чисел не больших  $x$ . По замечанию выше  $T\pi(\sqrt[5]{n}) < 100000 \cdot 550 = 55 \cdot 10^6$ . То есть алгоритм с такой сложностью вполне уложится по времени.

## День седьмой (18.02.2011 г.)

### Конкурс Эльдара Богданова и Ивана Метельского

#### Об авторах...

**Богданов Эльдар Фаикович**, родился в 1988 году в городе Тбилиси. Окончил Тбилисский Государственный Университет со степенью бакалавра компьютерных наук. Занимался программированием в учебном центре "Мзиури". В 2009 году окончил Тбилисский Государственный Университет со степенью бакалавра компьютерных наук. С 2010 года — тренер команд Тбилисского Государственного Университета по спортивному программированию.



#### Основные достижения:

- призер республиканской Олимпиады школьников по информатике 2005 года;
- третий призер Открытого чемпионата Москвы 2008 года, Кубка Векуа 2008 и Открытого чемпионата Украины 2007 и 2009 года;
- второй призер Открытого чемпионата Южного Кавказа 2007 года, Олимпиады имени Лебедева и Глушкова 2008 года и Зимней Школы по программированию 2009 года;
- полуфиналист Google Code Jam 2008 года;
- член команды "Tbilisi SU Triumvirate", которая является первой командой Грузии, ставшей призёром Чемпионата Мира: 11-е место и бронзовые медали на Чемпионате Мира ACM ICPC 2009 года.

**Метельский Иван Сергеевич**, родился в 1985 году в городке Гороховце в России, в 2001 году окончил гимназию №1 города Слуцка, в 2007 году — Факультет Прикладной Математики и Информатики Белорусского Государственного университета.

В 2004 — 2007 гг. — координатор по задачам проекта “Test-the-Best”.

С 2007-го года по настоящее время — координатор по задачам алгоритмических соревнований, проводимых компанией TopCoder. Многократный лауреат премии специального фонда президента по социальной поддержке учащихся и студентов.

С 2010-го года — также координатор марафонских соревнований. Один из сотрудников Topcoder Дэвид Мэссинджер, рассказывая об уровне белорусских команд, отметил, что “уровень белорусских “topcoder-ов”, таких профессионалов как он — очень высок”.



Основные достижения:

- медалист олимпиад IOI'99, IOI'00, IOI'01;
- финалист ACM ICPC 2003;
- золотой медалист ACM ICPC 2004;
- участник финальных соревнований TCO 2005, GCJ Europe 2006, GCJ 2008, 2010.

## **Теоретический материал. Ориентированные графы. Задача 2-SAT.**

### **Глава 1 Базовые алгоритмы.**

#### **Часть 1.1 Поиск в глубину.**

Данная лекция посвящена некоторым алгоритмам на ориентированных графах. Наличие дуги из вершины  $a$  в вершину  $b$  будем обозначать как  $a \rightarrow b$ . Наличие пути из вершины  $a$  в вершину  $b$  (достижимость  $b$  из  $a$ ) будем обозначать как  $a \rightsquigarrow b$ .

Поиск в глубину — это процедура систематического обхода всех вершин графа. Обход в глубину обладает множеством полезных свойств, поэтому на его базе часто строятся алгоритмы решения различных задач на (ориентированных и неориентированных) графах. В частности, алгоритм Тарьяна выделения компонент сильной связности, рассмотренный в главе 2, базируется на поиске в глубину.

Процедура обхода в глубину из вершины  $v$  проста и интуитивно понятна. Мы рассматриваем все дуги  $v \rightarrow w$  и для каждой из них, если вершина  $w$  еще не была посещена в процессе обхода, рекурсивно запускаем из нее обход в глубину. Схема эта, в принципе, похожа на алгоритм обхода в ширину, но разница между этими двумя алгоритмами огромная: обход в ширину пытается “одновременно” обойти все вершины  $w$  для которых есть дуга  $v \rightarrow w$ , в то время как обход в глубину делает это последовательно. В результате эти два вида обхода имеют совершенно различные свойства, а сферы их применения почти не пересекаются.

Рассмотрим псевдокод обхода графа в глубину:

**Процедура DFS(вершина  $v$ )**

Пометить  $v$  как посещенную

**Для** всех вершин  $w$ ,  $v \rightarrow w$ :

**Если**  $w$  не посещена, **то**:

    DFS( $w$ )

Конец процедуры

Пометить все вершины как непосещенные.

**Для** всех вершин графа  $v$ :

**Если**  $v$  не посещена:

    DFS( $v$ )

Если во время просмотра дуги  $v \rightarrow w$  вершина  $w$  еще не посещена, то происходит вызов DFS( $w$ ). В этом случае будем говорить, что обход в глубину проходит дугу  $v \rightarrow w$ . Можно показать, что граф, состоящий из всех пройденных дуг, является ориентированным лесом. В связи с этим пройденные дуги также называют *дугами дерева*.

После того, как обход в глубину завершен, все дуги графа можно разделить на 4 класса:

1. Дуги дерева, определенные выше.
2. Прямые дуги. Дуга  $v \rightarrow w$  называется *прямой*, если  $v$  и  $w$  лежат в одном дереве поиска в глубину, и  $w$  является потомком  $v$ , но не сыном  $v$ .
3. Обратные дуги. Дуга  $v \rightarrow w$  называется *обратной*, если  $v$  и  $w$  лежат в одном дереве поиска в глубину, и  $w$  является предком  $v$ .
4. Перекрестные дуги. Дуга  $v \rightarrow w$  называется *перекрестной*, если

она не принадлежит ни одному из трех предыдущих классов. Обратите внимание, что возможны два типа перекрестных дуг: между вершинами одного и того же дерева и между вершинами двух различных деревьев.

Отметим одно простое свойство, которое понадобится при обосновании корректности алгоритма Тарьяна.

**Лемма 1.1.** Если  $a \rightarrow b$  - перекрестная дуга, то вызов  $\text{DFS}(b)$  происходит раньше вызова  $\text{DFS}(a)$ .

**Доказательство.** Предположим, от противного, что вызов  $\text{DFS}(b)$  происходит после вызова  $\text{DFS}(a)$ . Рассмотрим момент, когда поиск в глубину рассматривает дугу  $a \rightarrow b$ . Если вызов  $\text{DFS}(b)$  еще не произошел, то вершина  $b$  еще не посещена, следовательно, дуга  $a \rightarrow b$  станет дугой дерева, стало быть, вызов  $\text{DFS}(b)$  уже произошел.

Так как вызов  $\text{DFS}(a)$  еще не закончил работу, то вызов  $\text{DFS}(b)$  произошел в процессе работы  $\text{DFS}(a)$ . Нетрудно показать, что все вершины, посещаемые в процессе вызова  $\text{DFS}(a)$ , становятся потомками вершины  $a$  в дереве поиска в глубину. Таким образом,  $b$  - потомок  $a$ , что означает, что дуга  $a \rightarrow b$  прямая. Противоречие.

## Часть 2.2 Топологическая сортировка.

Топологическим упорядочиванием вершин ориентированного графа  $G$  называется такая последовательность  $S$  всех вершин графа, что для любой дуги  $u \rightarrow v$  графа  $G$  вершина  $u$  расположена в  $S$  раньше, чем вершина  $v$ .

Введенное понятие имеет простую геометрическую интерпретацию. Если расположить вершины графа на прямой слева направо в точности в том же порядке, в котором они следуют в  $S$ , то любая дуга графа будет ориентирована слева направо.

Топологическое упорядочивание может быть построено не для любого графа. В частности, из геометрической интерпретации легко видеть, что если в графе есть цикл, то его топологическое упорядочение невозможно. В самом деле, предположим, что граф с циклом имеет топологическое упорядочение. Начиная с произвольной вершины цикла, начнем последовательно обходить цикл по дугам графа. Каждый переход по дуге ведет нас правее и правее от исходной вершины, что противоречит тому, что на некоторой этапе мы должны в нее вернуться (в силу замкнутости цикла).

Предположим, что граф  $G$  является ациклическим, и попробуем построить для него топологический порядок. Рассмотрим самую левую вершину в этом порядке. Так как все дуги идут слева направо, то в самую левую вершину никакая дуга входить не может. Покажем, что в ациклическом графе такая вершина всегда существует.

**Лемма 1.2.** Пусть  $G$  - ациклический граф. Тогда найдется вершина  $v$ , в которую не входит ни одна дуга.

**Доказательство.**

Пусть, от противного, в каждую вершину графа входит хотя бы одна дуга. Для произвольной вершины  $a$  обозначим любую из таких дуг  $next(a) \rightarrow a$ . Рассмотрим бесконечную последовательность  $(a, next(a), next(next(a)), next^3(a), next^4(a), \dots)$ . В силу ее бесконечности и конечности числа вершин в графе, какая-то вершина повторяется в ней дважды, например, вершина  $b$ . Рассмотрим часть между двумя повторениями:  $(b, next(b), next^2(b), \dots, b)$ . Развернув ее в противоположном направлении, получим цикл в графе. Противоречие.

Таким образом, в ациклическом графе всегда есть вершина без входящих дуг. Выберем любую из таких вершин и расположим первой в топологическом порядке. После этого все дуги из нее гарантированно идут слева направо, поэтому мы можем удалить выбранную вершину и все дуги из нее. Получим ациклический граф на единицу меньшим количеством вершин. В этом графе снова можно выбрать вершину без входящих дуг и расположить ее второй в топологическом порядке, и так далее. Повторив этот процесс  $V$  раз, мы построим в исходном графе топологический порядок.

Таким образом, доказана следующая теорема.

**Теорема 1.1.** Топологическое упорядочивание в графе  $G$  возможно тогда и только тогда, когда он не содержит циклов.

На базе подхода описанного выше, несложно построить линейный алгоритм топологической сортировки. Сначала этот алгоритм вычисляет входящую степень каждой вершины и помещает в очередь все вершины с нулевой степенью в очередь. Далее, вершины последовательно достаются из очереди и добавляются в конец строящегося топологического порядка. Дополнительно, для каждой вершины  $v$ , которую мы достаем из очереди, моделируется удаление всех выходящих из нее дуг  $v \rightarrow w$ . Другими словами, для каждой такой дуги входящая степень вершины  $w$  уменьшается на единицу. Если после этого уменьшения входящая степень стала равна нулю, вершина  $w$  помещается в очередь. Алгоритм работает, пока очередь не станет пустой. Если в итоге все вершины были помещены в очередь, то построен топологический порядок, в противном случае после удаления некоторых вершин мы получили граф, в котором нет вершин входящей степени ноль. По лемме 1.1 это возможно, только если в графе есть цикл, следовательно, топологического упорядочивания в этом графе не существует.

Псевдокод описанного алгоритма приведен ниже:

`inDegree` – массив входящих степеней вершин,  
изначально заполненный нулями

**Для** дуг графа  $v \rightarrow w$ :

`inDegree[w] := inDegree[w] + 1`

**Для** всех вершин графа  $v$ :

**Если** `inDegree[v] = 0`:

Добавить  $v$  в очередь

Изначально топологический порядок пуст

**Пока** очередь не пуста:

Достать очередную вершину  $v$  из очереди

Поместить  $v$  в конец топологического порядка

**Для** всех дуг  $v \rightarrow w$ :

`inDegree[w] := inDegree[w] - 1`

**Если** `inDegree[w] = 0`:

Добавить вершину  $w$  в очередь

**Если** все вершины были добавлены в очередь:

Построен топологический порядок.

**иначе**:

Топологическое упорядочивание не существует.

Нетрудно видеть, что как время работы, так и затраты памяти описанного алгоритма линейны.

## Глава 2 Компоненты сильной связности.

### Часть 2.1 Определение и базовые свойства.

Пусть  $G$  – ориентированный граф. *Компонентой сильной связности* в этом графе называется максимальное по включению подмножество его вершин такое, что любые две вершины из этого подмножества достижимы друг из друга.

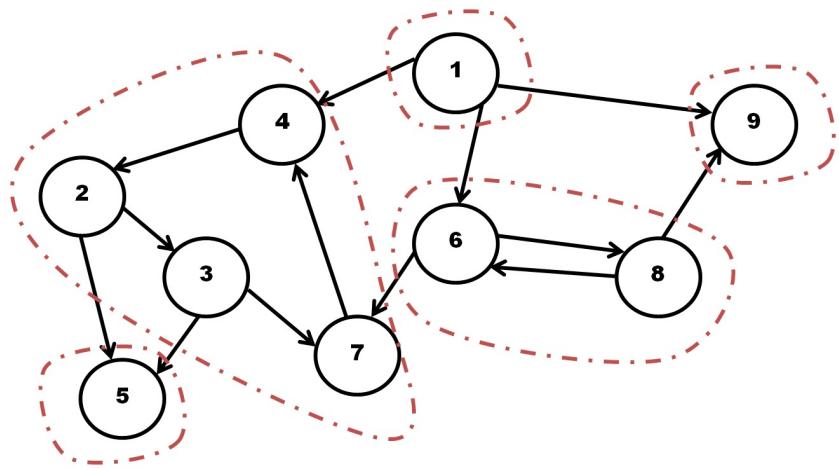


Рисунок 2.1 – Ориентированный граф  $G$  с выделенными компонентами сильной связности

Здесь и далее в этой главе “ $b$  достижима из  $a$ ” будет обозначаться как  $a \sim > b$ .

**Лемма 2.1** Компоненты сильной связности образуют разбиение всех вершин графа.

**Доказательство.**

Любая вершина графа  $x$  лежит хотя бы в одной компоненте сильной связности. В самом деле, можно рассмотреть множество  $\{x\}$ . Любые две его вершины достижимы друг из друга, поэтому его можно расширить до компоненты сильной связности.

Покажем, что любая вершина графа  $x$  лежит ровно в одной компоненте сильной связности. Пусть, от противного, она лежит в двух различных компонентах связности,  $A$  и  $B$ . По определению, ни одна из компонент не является подмножеством другой.

Рассмотрим произвольную вершину  $y$  из  $A \setminus B$  и произвольную вершину  $z$  из  $B \setminus A$ . Так как вершины  $x$  и  $y$  лежат в  $A$ , имеем  $x \sim > y$  и  $y \sim > x$ . Аналогично, так как  $x$  и  $z$  лежат в  $B$ , имеем  $x \sim > z$  и  $z \sim > x$ . Отсюда  $y \sim > x \sim > z$  и  $z \sim > x \sim > y$ , т.е. вершины  $y$  и  $z$  достижимы друг из друга. Таким образом, объединение  $A$  и  $B$  также является компонентой сильной связности, строго включающей в себя как  $A$ , так и  $B$ . Это противоречит максимальности  $A$  и  $B$  по включению.

По ориентированному графу  $G$  построим неориентированный граф  $G'$ , имеющий те же вершины. Будем проводить ребро  $a - b$  в  $G'$  тогда и только тогда, когда в  $G$  имеет место  $a \sim > b$  и  $b \sim > a$ .

**Лемма 2.2** Компоненты связности графа  $G'$  являются кликами.

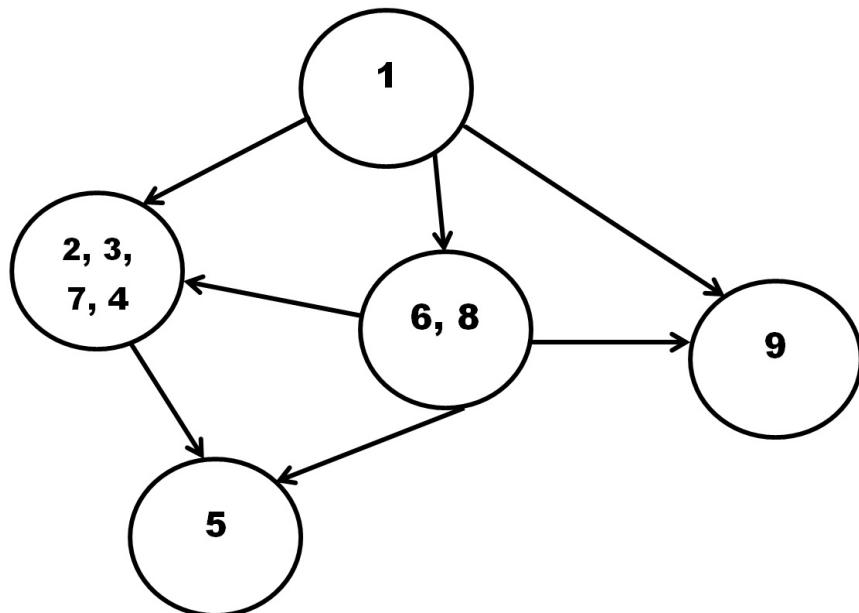
**Доказательство.**

Обозначим  $a \sim b$  условие “ $a \sim> b$  и  $b \sim> a$ ”. Легко показать, что  $\sim>$  является отношением эквивалентности, откуда сразу следует утверждение леммы.

Следствие 2.1 Компоненты связности графа  $G'$  соответствуют компонентам сильной связности графа  $G$ .

Следствие 2.2 Любой цикл в графе  $G$  никогда не выходит за пределы одной компоненты сильной связности.

С компонентами сильной связности тесно связано понятие *конденсации* графа. Пусть исходный граф имеет  $k$  компонент сильной связности  $C_1, C_2, \dots, C_k$ . Тогда в графе конденсации будет  $k$  вершин — по одной для каждой компоненты связности. Будем отождествлять вершину графа конденсации с соответствующей компонентой связности, т.е. вершины графа конденсации также обозначаются  $C_i$ . Дуга  $C_i \rightarrow C_j$  в графе конденсации проводится тогда и только тогда, когда существуют вершина  $x$  из  $C_i$  и вершина  $y$  из  $C_j$  такие, что  $x \rightarrow y$  является дугой исходного графа. Другими словами, граф конденсации получается из исходного ориентированного графа посредством “сжатия” каждой компоненты связности в одну вершину.

Рисунок 2.2 – Конденсация графа  $G$ 

Важнейшим свойством графа конденсации является то, что он ацикличен. В самом деле, предположим от противного, что в графе конденсации есть цикл. Рассмотрим дуги, соответствующие этому циклу в исходном графе (если некоторым дугам в конденсации соответствует более одной дуги в

исходном графе, выберем произвольную из них). Для каждой компоненты из цикла, в нее входит одна дуга, и из нее выходит одна дуга. Построим в каждой компоненте путь из “точки входа” в нее в “точку выхода” (это всегда возможно, так как в компоненте сильной связности из любой вершины достижима любая). Построенные пути в совокупности с дугами между компонентами сильной связности образуют цикл в исходном графе. Этот цикл проходит более чем по одной компоненте сильной связности, что противоречит лемме 2.2.

При решении задач граф конденсации оказывается полезен как раз за счет своей ацикличности. Многие задачи решаются на произвольных графах гораздо сложнее, чем на ациклических. Поэтому попытка сведения задачи на произвольном графе к точно такой же (или какой-то другой) задаче на графике конденсации является полезным приемом решения задач на ориентированных графах.

### **Часть 2.2 Простые алгоритмы: $O(V^3)$ и $O(V(V + E))$ .**

Существуют алгоритмы выделения компонент сильной связности, работающие за линейное время  $O(V + E)$ . Однако, в случаях когда исходный график невелик и хочется сэкономить немного времени при реализации, стоит применить более медленный, но при этом более простой алгоритм. В этой части мы рассмотрим два таких алгоритма.

Алгоритм, работающий за  $O(V^3)$ , основан на построении транзитивного замыкания исходного графа при помощи алгоритма Флойда-Уоршалла. Как только оно построено, мы можем построить график  $G'$  из леммы 2.2 и найти в нем компоненты связности. С учетом того, что компоненты связности являются кликами, можно поступить еще проще, заметив, что компонента связности, включающая в себя заданную вершину  $v$ , будет содержать кроме самой вершины  $v$  в точности те вершины, которые соединены с  $v$  ребром в графике  $G'$ .

Псевдокод описанного алгоритма приведен ниже:

```
// Исходные данные:  
// массив  $A[1..N, 1..N]$  такой что  
//  $A[i, j] = \text{true}$  тогда и только тогда,  
// когда  $i \rightarrow j$  --- дуга исходного графа  
// Алгоритм Флойда-Уоршалла  
Для  $i$  от 1 до  $N$ :  
     $A[i, i] := \text{true}$   
Для  $k$  от 1 до  $N$ :  
    Для  $i$  от 1 до  $N$ :  
        Для  $j$  от 1 до  $N$ :  
             $A[i, j] |= A[i, k] \& A[k, j]$ 
```

```
// компоненты сильной связности
comprCnt := 0
comprId := (-1, -1, ..., -1) [N элементов]

Для i от 1 до N:
    Если comprId[i] = -1:
        Для j от 1 до N:
            Если A[i, j] && A[j, i]:
                comprId[j] := comprCnt
                comprCnt := comprCnt + 1

// результат работы:
// comprCnt --- общее количество компонент
// comprId[i] --- идентификатор компоненты, содержащей вершину i
```

Алгоритм, работающий за  $O(V(V + E))$ , основан на аналогичной идеи, но достижимость в нем проверяется посредством поиска в глубину. Построим граф  $G_{rev}$ , содержащий те же вершины что и исходный граф  $G$ , и те же дуги, но в обратном направлении, т.е. дуга  $x \rightarrow y$  графа  $G$  преобразуется в дугу  $y \rightarrow x$  графа  $G_{rev}$ . Теперь для того, чтобы выделить компоненту сильной связности, содержащую заданную вершину  $v$ , запустим поиск в глубину из вершины  $v$  в каждом из графов  $G$  и  $G_{rev}$ . Если вершина  $w$  в обоих случаях оказалась посещена, то имеем  $v \sim> w$  и  $w \sim> v$ . Следовательно, компонента сильной связности состоит в точности из вершин, которые были посещены в каждом из этих поисков в глубину. Получили алгоритм, затрачивающий время  $O(V + E)$  на каждую компоненту сильной связности, следовательно, общее время его работы есть  $O(V(V + E))$ .

### Часть 2.3 Алгоритм Тарьяна.

Рассмотрим в графе  $G$  произвольную вершину  $v_0$  и запустим поиск в глубину из этой вершины. Нетрудно видеть, что в процесс поиска в глубину будут посещены все вершины всех компонент сильной связности, достижимых из вершины  $v_0$ .

Логично предположить, что если мы рассмотрим, как расположены вершины одной сильно связной компоненты в дереве поиска в глубину, то это расположение вряд ли может быть совершенно произвольным. Попробуем установить, какими свойствами оно обладает.

Пусть  $C$  — некоторая компонента сильной связности, и  $r(C)$  — ее вершина, имеющая минимальную высоту в дереве поиска в глубину (если таких вершин несколько, выберем ту из них, которая посещается первой в процессе поиска в глубину). По определению,  $r(C)$  — это первая из вершин компоненты  $C$ , которая будет посещена в процессе поиска в глубину. Так как все остальные вершины компоненты достижимы из  $r(C)$ , то все

они попадут в поддерево дерева поиска в глубину с корнем в  $r(C)$ . В связи с этим вершина  $r(C)$  называется корнем компоненты  $C$ .

Следующее важное свойство состоит в следующем: если на каком-то шаге поиска в глубину мы покинули компоненту сильной связности  $C$ , то в рамках текущего поддерева дерева поиска в глубину мы в нее уже не вернемся.

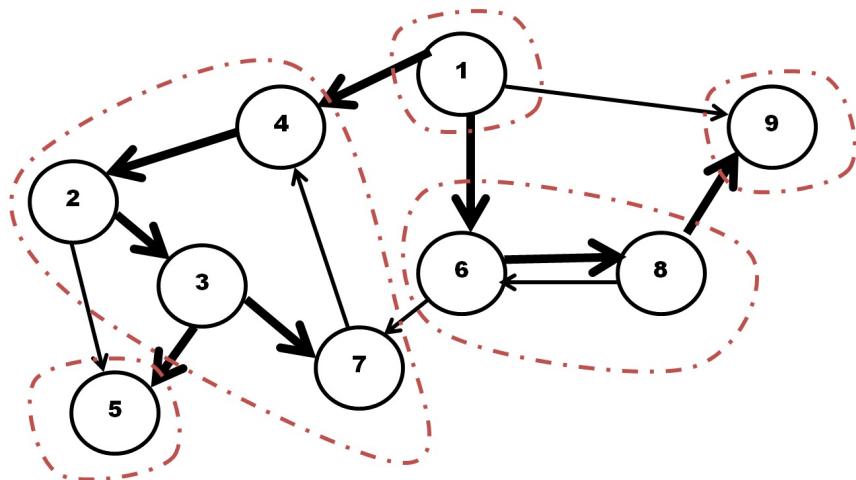


Рисунок 2.3 – Обход в глубину графа  $G$

**Лемма 2.3** Пусть  $v \rightarrow w$  — дуга дерева поиска в глубину, причем  $v$  принадлежит компоненте сильной связности  $A$ , а  $w$  — компоненте сильной связности  $B$  ( $A \neq B$ ). Тогда в поддереве дерева поиска в глубину с корнем в  $w$  нет вершин из  $A$ .

#### Доказательство.

Пусть, от противного, некоторая вершина  $x$  из поддерева с корнем в  $w$  принадлежит компоненте  $A$ . Тогда мы имеем  $v \rightarrow w \sim> x \sim> v$ , откуда, по лемме 2,  $w$  также должно принадлежать компоненте  $A$ . Противоречие.

Предположим, что мы для каждой вершины умеем определять, является она корнем некоторой компоненты сильной связности или нет. Тогда, основываясь на сформулированных свойствах расположения компонент сильной связности в дереве поиска в глубину, мы уже готовы представить линейный алгоритм выделения компонент сильной связности.

Псевдокод его приведен ниже:

**Глобальные переменные:**

Стек вершин  $S$

Массив пометок, посещена вершина или нет

**Процедура**  $DFS(\text{вершина } v)$

Добавить в стек  $S$

Пометить  $v$  как посещенную

**Для** всех вершин  $w$ ,  $v \rightarrow w$ :

**Если**  $w$  непосещена, то:

$DFS(w)$

**Если**  $v$  является корнем компоненты сильной связности:

Доставать из  $S$  вершины, пока не достанем  $S$ .

Все взятые из  $S$  вершины образуют

новую компоненту сильной связности.

Конец процедуры

$S :=$  пустой стек

Пометить все вершины как не посещенные.

Для всех вершин графа  $v$ :

**Если**  $v$  не посещена:

$DFS(v)$

Покажем, что приведенный алгоритм работает корректно, т.е., что из стека всякий раз достаются в точности вершины компоненты сильной связности с корнем в  $v$  (обозначим эту компоненту через  $A$ ). Это можно доказать индукцией по дереву — от листьев к корню. Мы начинаем доставать вершины компоненты  $A$  из стека в тот момент, когда обход поддерева с корнем  $v$  закончен, в частности, все корни компонент сильной связности, являющиеся потомками  $v$ , уже посещены, и, следовательно, по индуктивному предположению, эти компоненты уже правильно выделены. Значит, в стеке после вершины  $v$  находятся все вершины компоненты сильной связности с корнем  $v$ , и, возможно, еще вершины некоторых других компонент сильной связности. Пусть  $B$  — одна из таких компонент сильной связности. Корень  $B$  должен быть предком вершины  $v$  (поскольку все компоненты с корнями-потомками уже выделены). С момента, когда в процессе поиска в глубину мы вошли в корень компоненты  $B$  до момента, когда мы вошли в вершину  $v$ , мы в некоторый момент времени вышли из компоненты  $B$ . Тогда, по лемме 2.2, в соответствующем поддереве не может быть вершин из  $B$ , а, так как поддерево с корнем в  $v$  расположено внутри этого под дерева, в нем также не может быть вершин из  $B$ . Таким образом, вершин из других компонент сильной связности в стеке после вершины  $v$  быть не мо-

жет, поэтому компонента сильной связности с корнем в  $v$  будет выделена правильно.

Перейдем к рассмотрению того, как мы можем определить, является ли вершина  $v$  корнем некоторой компоненты сильной связности или нет. Для этого предлагается для каждой вершины  $v$  вычислить два значения:  $index(v)$  и  $lowlink(v)$ . Псевдокод их вычисления приведен ниже:

```
globalIndex := 0
Процедура DFS(вершина v)
    ...
    index(v) := globalIndex
    globalIndex := globalIndex + 1
    lowlink(v) := index(v)

    ...
    Для всех вершин w, v → w:
        Если w не посещена, то:
            ...
            lowlink(v) := min(lowlink(v), lowlink(w))
            иначе, если w находится в S:
                lowlink(v) := min(lowlink(v), index(w))
            ...
Конец процедуры
```

Смысл величины  $index(v)$  очень прост — это аналог времени входа в вершину  $v$ . Другими словами, чем раньше вершина  $v$  посещается в процессе поиска в глубину, тем меньше будет значение  $index(v)$ .

Смысл значения  $lowlink(v)$  менее очевиден. Несколько проясняет его следующая лемма.

**Лемма 2.4**  $lowlink(v) = index(w)$ , где  $w$  — некоторая вершина, лежащая в той же компоненте сильной связности, что и  $v$ .

#### Доказательство.

Лемму будем доказывать по индукции, от листьев дерева к корню.

Пусть для всех потомков вершины  $v$  утверждение леммы верно. Докажем, что оно верно и для самой вершины  $v$ .

В алгоритме есть несколько мест, где значение  $lowlink(v)$  изначально устанавливается или изменяется. Покажем, что во всех этих местах в результате  $lowlink(v)$  всегда будет равен  $index(w)$ , где  $w$  — некоторая вершина из той же компоненты сильной связности, что и  $v$ .

- Инициализация:  $lowlink(v) := index(v)$ . Здесь, очевидно, утверждение леммы верно.
- Пересчет №1: если  $w$  не посещена, то  $lowlink(v) := \min(lowlink(v), lowlink(w))$ . Здесь возможны два варианта. Если  $w$  лежит в той же самой компоненте сильной связности, то утверждение леммы верно по индуктивному предположению. Если же  $w$  лежит в другой компоненте сильной связности, то по индуктивному предположению  $lowlink(w)$  равен значению  $index$  для некоторой вершины этой компоненты. Но все вершины этой компоненты лежат в поддереве с корнем  $w$ , поэтому значения  $index$  для них всех строго больше  $index(v)$ , а значение  $lowlink(v)$ , в силу инициализации и способа последующего обновления, никогда не превосходит  $index(v)$ . Поэтому обновления  $lowlink(v)$  в этом случае не произойдет.
- Пересчет №2: если  $w$  находится в  $S$ , то  $lowlink(v) := \min(lowlink(v), index(w))$ . Заметим, что в  $S$  находятся вершины только тех компонент, которые на данный момент в процессе выделения. Это могут быть только компоненты, корень которых является предком  $v$ , так как все остальные компоненты либо уже полностью выделены, либо еще не начали выделяться. Пусть  $w$  находится в некоторой компоненте сильной связности  $A$ . Имеем  $w \sim r(A) \sim v \rightarrow w$ , откуда получаем, что  $v$  и  $w$  находятся в одной и той же компоненте сильной связности. Следовательно, утверждение леммы после этого обновления также остается в силе.

Таким образом, при инициализации и всех последующих обновлениях имеем  $lowlink(v) = index(w)$ , где  $w$  некоторая вершина из той же компоненты сильной связности, что и  $v$ .

Следствие 2.1 Пусть вершина  $v$  принадлежит компоненте сильной связности  $C$ . Если  $v = r(C)$ , то  $lowlink(v) = index(v)$ .

Доказательство.

С учетом инициализации и последующих обновлений, имеем  $lowlink(v) \leq index(v)$ . С другой стороны, по лемме 2.4,  $lowlink(v) = index(w)$ , где  $w$  — некоторая вершина из  $C$ . Для всех вершин  $w$  из  $C$  выполняется  $index(w) \geq index(r(C))$ , таким образом,  $lowlink(v) \geq index(v)$ . Комбинируя два полученных неравенства, имеем  $lowlink(v) = index(v)$ .

Итак, для корней сильно связных компонент  $lowlink(v)$  и  $index(v)$  совпадают. В то же время, в доказательствах выше было отмечено, что  $lowlink(v) \leq index(v)$ . Оказывается, что если вершина не является кор-

нем сильно связной компоненты, то это неравенство становится строгим:  $lowlink(v) < index(v)$ .

Чтобы показать это, нам понадобится вспомогательная лемма.

**Лемма 2.6** Пусть вершина  $v$  принадлежит компоненте сильной связности  $C$  и не является ее корнем. Тогда в графе существует путь  $(x_0 = v, x_1, \dots, x_k, x_{k+1})$ ,  $k \geq 0$ , такой, что все дуги  $x_0 \rightarrow x_1, x_1 \rightarrow x_2, \dots, x_{k-1} \rightarrow x_k$  являются дугами поиска в глубину,  $x_{k+1}$  принадлежит  $C$  и  $index(x_{k+1}) < index(v)$ .

**Доказательство.**

Будем говорить, что вершина  $w$  является нестрогим потомком вершины  $v$ , если  $w = v$  или  $w$  является потомком вершины  $v$  в дереве поиска в глубину.

Рассмотрим произвольный простой путь из  $v$  в  $r(C)$ :  $(y_0 = v, y_1, \dots, y_t = r(C))$ . Так как  $r(C)$  не является нестрогим потомком  $v$ , то найдется такое  $i > 0$ , что все вершины  $y_i, y_{i+1}, \dots, y_t$  не являются нестрогими потомками  $v$ , а  $y_{i-1}$  — нестрогий потомок  $v$ .

Раз  $y_{i-1}$  — нестрогий потомок  $v$ , то существует путь из  $v$  в  $y_{i-1}$ , состоящий только из дуг дерева поиска в глубину:  $(z_0 = v, \dots, z_q = y_{i-1})$ ,  $q \geq 0$ . Покажем, что путь  $(z_0, \dots, z_q, y_i)$  удовлетворяет условиям леммы.

Так как  $y_i \sim > r(C) \sim > v \sim > y_i$ , то  $y_i$  принадлежит  $C$ , поэтому осталось показать, что  $index(y_i) < index(v)$ . Так как  $y_i$  — не является нестрогим потомком  $v$ , то дуга  $z_q \rightarrow y_i$  может быть только обратной или перекрестной. Если это обратная дуга, то  $y_i$  должна быть предком  $v$ , что влечет  $index(y_i) < index(v)$ . Если же это перекрестная дуга, то  $index(y_i) < index(z_q)$ . Возможность  $index(v) \leq index(y_i) < index(z_q)$  исключена, т.к. в этом случае  $y_i$  была бы потомком  $v$ , поэтому мы снова имеем  $index(y_i) < index(v)$ . Таким образом, в обоих случаях  $index(y_i) < index(v)$ , что завершает доказательство леммы.

**Следствие 2.2** Пусть вершина  $v$  принадлежит компоненте сильной связности  $C$ . Если  $v \neq r(C)$ , то  $lowlink(v) < index(v)$ .

**Доказательство.**

Рассмотрим путь из леммы 2.6:  $(x_0 = v, x_1, \dots, x_k, x_{k+1})$ . Индукцией по  $i = k, k-1, \dots, 0$  несложно показать, что  $lowlink(x_i) \leq index(x_{k+1})$ .

База:  $i = k$ . При рассмотрении дуги  $x_k \rightarrow x_{k+1}$ , с учетом того, что  $x_{k+1}$  принадлежит  $C$  (а значит, находится в стеке  $S$ ), будет совершено обновление  $lowlink(x_k) := \min(lowlink(x_k), index(x_{k+1}))$ , откуда  $lowlink(x_k) \leq index(x_{k+1})$ .

Переход:  $i < k$ . При рассмотрении дуги  $x_i \rightarrow x_{i+1}$ , которая является дугой дерева поиска в глубину, будет совершено обновление  $lowlink(x_i) := \min(lowlink(x_i), lowlink(x_{i+1}))$ . По индукции,  $lowlink(x_{i+1}) \leq index(x_{k+1})$ , откуда  $lowlink(x_i) \leq index(x_{k+1})$ .

В результате получаем

$$\text{lowlink}(v) = \text{lowlink}(x_0) \leq \text{index}(x_{k+1}) < \text{index}(v).$$

Доказанные результаты дают нам простой и эффективный способ определения, является ли вершина корнем компоненты сильной связности — нужно всего лишь проверить равенство  $\text{lowlink}(v) = \text{index}(v)$ , оно будет выполняться тогда и только тогда, когда  $v$  — корень компоненты сильной связности.

На этом описание алгоритма Тарьяна завершено. Нетрудно видеть, что время его работы и затраты памяти линейны:  $O(V + E)$ .

Упражнение 2.1. Докажите следующее полезное с практической точки зрения свойство: алгоритм Тарьяна выделяет компоненты в порядке, соответствующем обратному топологическому порядку графа конденсации. Другими словами, если  $A \rightarrow B$  — дуга графа конденсации, то компонента  $B$  будет выделена до компоненты  $A$ .

## Глава 3 Задача 2-SAT.

### Часть 3.1 Постановка задачи.

Пусть  $x_1, x_2, \dots, x_n$  — булевы переменные, т.е. каждое  $x_i$  может принимать значение True или False.

Булевые формулы вида  $x_i$  или  $!x_i$  будем называть *литералами*. Таким образом, литерал — это либо одиночная переменная, либо отрицание одиночной переменной.

Формулу, представляющую собой дизъюнкцию одного или более литералов, будем называть *элементарной дизъюнкцией*. Например, такова формула  $x_1 \parallel !x_7 \parallel !x_{10} \parallel x_3$ .

Будем говорить, что формула записана в конъюнктивной нормальной форме (CNF), если она представляет собой конъюнкцию одной или более элементарных дизъюнкций. Например,  $(x_1 \parallel !x_7 \parallel !x_{10} \parallel x_3) \& \& (x_2 \parallel x_3) \& \& (!x_{1024})$ .

Если каждая из элементарных дизъюнкций формулы в CNF содержит не более  $k$  литералов, то говорят, что формула записана в  $k$ -CNF.

Рассмотрим следующую задачу.

**Задача  $k$ -SAT.** Задана булева функция, представленная формулой в виде  $k$ -CNF. Существует ли набор значений переменных, на котором заданная функция истинна?

К сожалению, задача  $k$ -SAT является NP-полной для любого  $k \geq 3$ . Для задачи же 2-SAT существует полиномиальный (более того, линейный) алгоритм решения, рассмотрению которого и посвящена данная глава.

### **Часть 3.2 Вспомогательные результаты.**

Рассмотрим формулу в 2-CNF. Для удобства будем считать, что каждая элементарная дизъюнкция в ней содержит ровно 2 литерала (т.к. формула  $a$  эквивалентна  $a \parallel a$ , то каждую дизъюнкцию с 1-м литералом легко поменять на дизъюнкцию с двумя).

Здесь и далее будем обозначать общее количество переменных через  $n$ , а общее количество элементарных дизъюнкций через  $m$ . Наша конечная цель — научиться решать задачу 2-SAT за время  $O(n + m)$ . Заметим, что элементарная дизъюнкция  $a \parallel b$  эквивалентна формуле  $(!a \geq b) \&\& (!b \geq a)$ . Это легко проверить непосредственной подстановкой.

На базе этого построим ориентированный граф, содержащий  $2n$  вершин и  $2m$  дуг. Каждому из  $2n$  возможных литералов над  $n$  переменными в графе будет соответствовать ровно одна вершина. Далее в тексте, говоря о  $x_i$  или  $!x_i$ , мы можем иметь в виду как литерал, так и соответствующую ему вершину графа.

Дуги графа строятся следующим образом: для каждой элементарной дизъюнкции  $a \parallel b$ , где  $a$  и  $b$  — литералы, добавим в граф две дуги: из  $!a$  в  $b$  и из  $!b$  в  $a$  (легко увидеть, что отрицание литерала также является литералом).

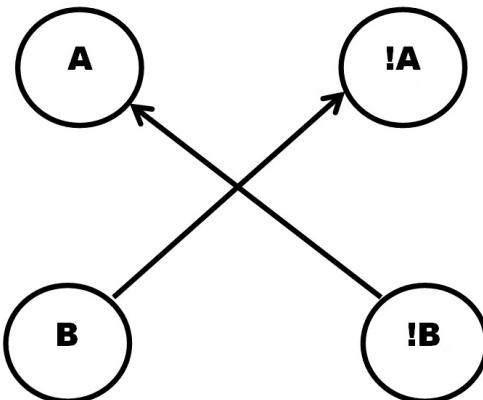
Заметим, что построенный граф обладает интересным свойством, которое в дальнейшем окажется очень полезным для обоснования корректности рассматриваемых алгоритмов решения задачи 2-SAT.

**Лемма 3.1.** В построенном графе путь из  $a$  в  $b$  существует тогда и только тогда, когда в нем существует путь из  $!b$  в  $!a$ .

#### **Доказательство.**

По построению, дуга из  $a$  в  $b$  в графе существует тогда и только тогда, когда в нем есть дуга из  $!b$  в  $!a$ . Отсюда, применяя индукцию по количеству дуг в пути, получаем утверждение леммы.

Рассмотрим соответствие между наборами значений переменных  $x_i$  и множествами из  $n$  выделенных вершин построенного графа, которое определяется следующим образом: если  $x_i = 1$ , то выделим вершину  $x_i$ , если же  $x_i = 0$ , то выделим вершину  $!x_i$ . Очевидно, что при данном соответствии для каждого  $i$  ровно одна из вершин  $x_i$  и  $!x_i$  будет выделена, и, если ограничиться только множествами из выделенных вершин, удовлетворяющими этому ограничению, то получим взаимно однозначное соответствие. Далее, говоря о выделенном множестве вершин, будем иметь в виду, что



ровно одна из вершин  $x_i$  и  $\neg x_i$  выделена.

От построенного графа будет мало пользы, если мы не сформулируем в терминах этого графа простое условие, при котором множество выделенных вершин соответствует набору значений переменных, на котором исходная функция истинна.

Будем говорить, что множество выделенных вершин замкнуто относительно достижимости, если выполнено следующее условие: для любых вершин  $a$  и  $b$  таких, что  $b$  достижима из  $a$ , если вершина  $a$  выделена, то и вершина  $b$  выделена.

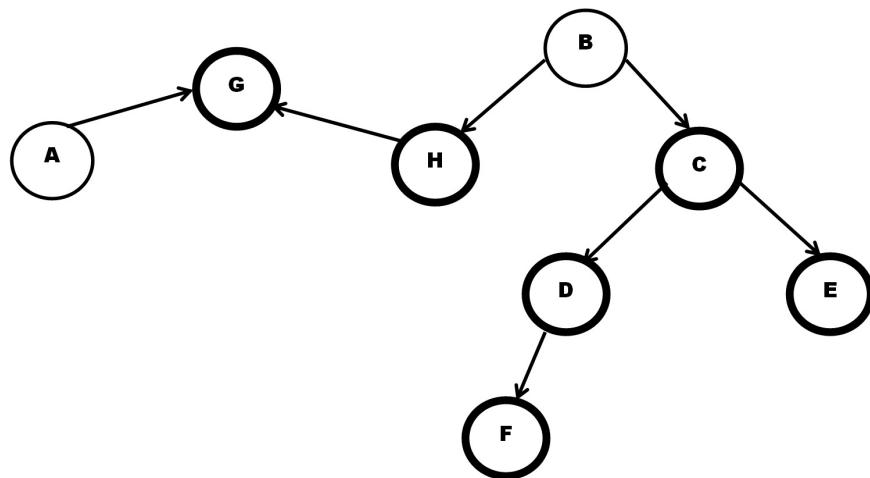


Рисунок 2.6 – Множество, замкнутое относительно достижимости

**Лемма 3.2** Множество выделенных вершин соответствует набору значений переменных, на котором исходная функция истинна, тогда и только тогда, когда оно замкнуто относительно достижимости.

#### Доказательство.

Прежде всего, заметим, что условие замкнутости относительно достижимости эквивалентно следующему ослабленному его варианту: для любых вершин  $a$  и  $b$  таких, что в графе есть дуга из  $a$  в  $b$ , если вершина  $a$  выделена, то и вершина  $b$  выделена. В самом деле, из замкнутости относительно достижимости сразу же следует выполнение ослабленного условия, а из выполнения ослабленного условия замкнутость по достижимости легко показывается индукцией по длине пути из  $a$  в  $b$ .

Формула в 2-CNF истинна тогда и только тогда, когда каждая ее элементарная дизъюнкция истинна. Рассмотрим элементарную дизъюнкцию  $a \parallel b$ . Если вершина  $\neg a$  выделена, а вершина  $b$  — нет, значит выделена вершина  $\neg b$  и дизъюнкция  $a \parallel b$  на соответствующем наборе переменных ложна. Аналогично, если вершина  $\neg b$  выделена, а вершина  $a$  — нет, то выделена вершина  $\neg a$ , и дизъюнкция снова ложна. Таким образом, для истинности

дизъюнкции  $a \parallel b$  необходимо выполнение ослабленного условия для каждой из дуг  $!a \rightarrow b$  и  $!b \rightarrow a$ . Нетрудно видеть, что обратное также верно: из выполнения ослабленного условия для каждой из дуг  $!a \rightarrow b$  и  $!b \rightarrow a$  следует истинность дизъюнкции  $a \parallel b$ . Отсюда получаем, что все дизъюнкции  $a \parallel b$  будут истинными тогда и только тогда, когда ослабленное условие выполнено для всех дуг графа, что в свою очередь будет тогда и только тогда, когда множество замкнуто относительно достижимости.

Таким образом, задача 2-SAT сводится к задаче о наличии в ориентированном графе определенного рода множества выделенных вершин, замкнутого относительно достижимости. В таком виде мы и будем решать задачу в следующих двух частях лекции.

### Часть 3.3 Полиномиальный алгоритм.

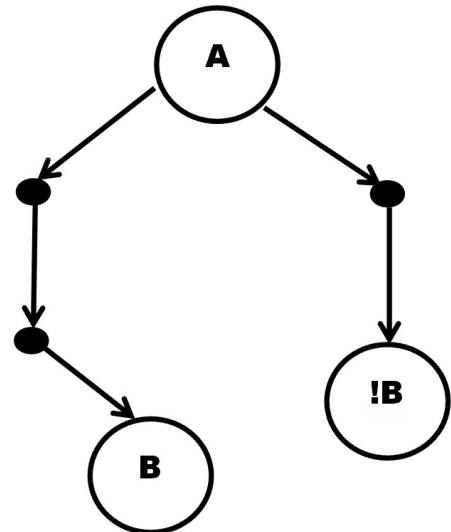
Попробуем разработать полиномиальный алгоритм на базе того, что результирующее множество выделенных вершин должно быть замкнуто относительно достижимости.

Назовем вершину  $a$  противоречивой, если найдется вершина  $b$  такая, что из  $a$  достижимы, как вершина  $b$ , так и вершина  $!b$ . При построении замкнутого относительно достижимости множества выделение вершины  $a$  влечет за собой выделение обеих вершин  $b$  и  $!b$ , но так как обе эти вершины одновременно выделять запрещено, то в результирующее множество выделенных вершин противоречивые вершины входить не могут.

Пусть изначально ни одна из вершин графа не выделена. Рассмотрим переменную  $x_1$ . Нам необходимо выделить одну из вершин  $x_1$  и  $!x_1$ . После того, как одна из них выделена, все вершины, достижимые из нее, также следует выделить, чтобы обеспечить замкнутость относительно достижимости.

Принципиально возможны три ситуации:

- *Обе вершины  $x_1$  и  $!x_1$  противоречивы.* Тогда ни одну из них нельзя выделить, поэтому решения не существует.
- *Ровно одна из вершин  $x_1$  и  $!x_1$  противоречива.* Тогда, очевидно, необходимо выделить вторую из них.
- *Ни одна из вершин  $x_1$  и  $!x_1$  не противоречива.* Здесь мы стоим перед выбором — какую из вершин выделить. Этого выбора хотелось бы



избежать, так как в случае, когда выбор придется делать для многих переменных, результирующий алгоритм может получиться экспоненциальным.

Интересный способ избегать выбора — делать его произвольным образом. Еще более интересно, что в данном конкретном случае такой подход приводит к правильному алгоритму.

Чтобы понять, почему это так, нужно сначала разобраться, каковы последствия выделения той или иной вершины в графе. Если оба способа выбора приводят к одним и тем же последствиям, то, очевидно, никакой разницы между ними, с точки зрения дальнейшей работы алгоритма, нет.

Пусть  $S$  — некоторое множество выделенных вершин в графе такое, что для каждого  $i$  не более одной из вершин  $x_i$  и  $\neg x_i$  выделено. Будем считать, что  $S$  — это промежуточный результат работы нашего алгоритма. Пусть  $a$  — некоторая вершина, такая, что ни  $a$ , ни  $\neg a$  не принадлежат  $S$ . Назовем эту вершину противоречивой относительно  $S$ , если найдется такая вершина  $b$ , что существует путь из  $a$  в  $b$  и  $\neg b$  принадлежит  $S$ . Так как выделение вершины  $a$  влечет за собой выделение  $b$ , а одновременное выделение  $b$  и  $\neg b$  невозможно, получаем, что вершина  $a$  не может быть добавлена к  $S$  с сохранением замкнутости относительно достижимости.

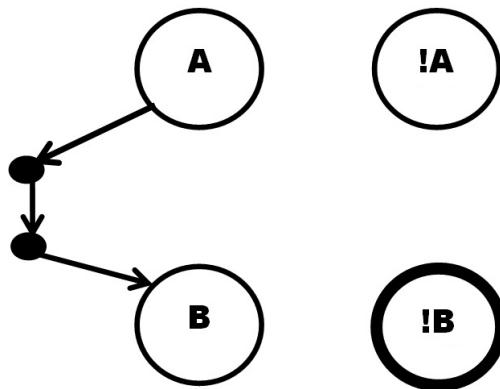


Рисунок 2.7 – Вершина  $A$  противоречива относительно  $\{\neg B\}$

Понятие противоречивости относительно множества выделенных вершин как раз и описывает последствия выбора. Дело в том, что если некоторая вершина  $a$  не противоречива, но противоречива относительно текущего множества выделенных вершин, то выделять эту вершину нельзя не потому, что ее вообще принципиально нельзя выделять (как в случае, если бы она была противоречива), а потому, что мы уже выделили некоторые вершины, при этом, возможно, принял некоторые решения. Если бы решения были приняты другим образом, вполне возможно, что вершина  $a$  не

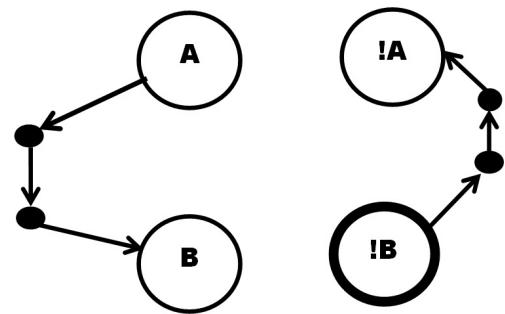
была бы противоречива относительно полученного множества выделенных вершин, и это как раз привело бы к нахождению решения задачи.

Казалось бы, мы вскрыли некоторую проблему, которая ставит концепцию произвольного выбора под сомнение. Пожалуй, такая концепция действительно была бы сомнительной для произвольного графа, однако наш граф обладает замечательным свойством из Леммы 3.1. На его базе несложно разрешить возникшую проблему.

**Лемма 3.3** Пусть  $S$  — множество выделенных вершин, замкнутое относительно достижимости. Тогда никакая вершина не является противоречивой относительно  $S$ .

### Доказательство.

От противного, пусть вершина  $a$  — противоречива относительно  $S$ . Это значит, что ни  $a$ , ни  $!a$  не принадлежат  $S$ , и существует вершина  $b$  такая, что  $b$  достижимо из  $a$ , и  $!b$  принадлежит  $S$ . Из леммы 3.1 и наличия пути из  $a$  в  $b$  получаем наличие пути из  $!b$  в  $!a$ . Раз множество  $S$  замкнуто относительно достижимости, и  $!b$  принадлежит  $S$ , то  $!a$  также должно принадлежать  $S$ . Противоречие.



Если, выделяя некоторую вершину, мы также будем выделять все достижимые из нее вершины, то текущее множество выделенных вершин всегда будет замкнутым относительно достижимости. А тогда, по только что доказанной лемме, противоречивых вершин относительно текущего множества выделенных вершин не будет. Другими словами, какую бы вершину мы не выделили, никаких последствий для дальнейшей работы алгоритма это за собой повлечь не может.

После того, как алгоритм концептуально обрисован и его корректность формально обоснована, самое время записать его более точно. Приведем псевдокод алгоритма:

Изначально ни одна вершина не выделена.

Для  $i$  от 1 до  $N$ :

Если ни одна из вершин  $x_i$  и  $!x_i$  не выделена, то:

Если вершина  $x_i$  не противоречива, то:

Выделить вершину  $x_i$  и все достижимые из нее вершины.

иначе, если вершина  $!x_i$  не противоречива, то:

Выделить вершину  $!x_i$  и все достижимые из нее вершины.

иначе:

Решения не существует.

Для проверки, противоречива вершина или нет, достаточно запустить из нее поиск в глубину, что позволит найти все вершины, достижимые из нее. Аналогично, для выделения всех вершин, достижимых из заданной, также может быть применен поиск в глубину.

С учетом того, что время работы поиска в глубину составляет  $O(V + E)$ , получаем, что общая времененная сложность построенного алгоритма составляет  $O(n \cdot (n + m))$ .

Замечание. На практике достаточно сложно построить пример, на котором описанная оценка  $n \cdot (n + m)$  будет достигаться, поэтому алгоритм вполне может проходить тесты даже для таких  $n$  и  $m$ , для которых, согласно оценке, он, казалось бы, должен работать слишком медленно. Дело в том, что выделение одной вершины обычно влечет за собой выделение еще некоторых вершин (достижимых из нее), и их уже не надо будет проверять на противоречивость.

Саму проверку на противоречивость можно несколько ускорить следующим образом. Если в поиске в глубину мы пришли в некоторую вершину  $a$  такую, что  $a$  или  $!a$  уже выделены, то дальше рекурсию из этой вершины можно не запускать, поскольку противоречия найдено не будет. Эта оптимизация не улучшает оценку  $O(n \cdot (n + m))$ , но делает задачу автора по построению тестов против вашего решения еще более сложной.

Еще один трюк, который может помочь сдать задачу — это рассмотрение переменных во внешнем цикле алгоритма не в порядке  $x_1, x_2, \dots, x_N$ , а в случайном порядке. Построение теста против конкретного порядка, безусловно, значительно легче построения теста против большинства (или всех) возможных порядков.

Все высказывание вовсе не означает рекомендацию применять описанный алгоритм во всех задачах, без учета ограничений на  $n$  и  $m$ . Однако, с учетом того, что линейный алгоритм порядочно сложнее в реализации, попытка сдачи более медленного алгоритма перед написанием линейного может оказаться вполне оправданной. В большом количестве случаев, с учетом описанных выше факторов, медленный алгоритм все равно пройдет авторские тесты (возможно, против желания автора). Даже если медленный алгоритм не пройдет авторские тесты, почти весь написанный код понадобится при реализации линейного алгоритма, так что фактически вы просто сделаете одну лишнюю отправку.

### **Часть 3.4 Линейный алгоритм.**

Перейдем к рассмотрению алгоритма, который гарантированно работает за линейное время  $O(n + m)$ .

Начнем с выделения в графе сильно связных компонент и построения конденсации.

Заметим, что если хотя бы одна из вершин сильно связной компоненты выделена, то для достижения замкнутости относительно достижимости все остальные вершины также должны быть выделены. Таким образом, каждая сильно связная компонента должна быть либо полностью выделена, либо полностью не выделена.

Если для некоторого  $i$  вершины  $x_i$  и  $\neg x_i$  лежат в одной и той же компоненте связности, то выделить ровно одну из них не получится, поэтому задача не имеет решения. В дальнейших рассуждениях будем считать, что для любого  $i$  вершины  $x_i$  и  $\neg x_i$  лежат в разных компонентах связности.

Следующее наблюдение: если вершины  $a$  и  $b$  лежат в одной и той же сильно связной компоненте, то существуют пути из  $a$  в  $b$  и из  $b$  в  $a$ . Тогда по лемме 3.1 также существуют пути из  $\neg b$  в  $\neg a$  и из  $\neg a$  в  $\neg b$ , таким образом вершины  $\neg a$  и  $\neg b$  также лежат в одной и той же сильно связной компоненте. Отсюда следует, что все сильно связные компоненты можно разбить на пары  $(A, B)$  такие, что компонента  $B$  состоит в точности из отрицаний литералов из компоненты  $A$ .

Для каждой такой пары  $(A, B)$  можно ввести новую переменную  $y_j$  и считать, что компонента  $A$  соответствует литералу  $y_j$ , а компонента  $B$  — литералу  $\neg y_j$ . После этого задача на исходном графе оказывается сведенной к задаче на его конденсации: нужно выделить некоторые вершины так, чтобы для любого  $j$  ровно одна из вершин  $y_j$  и  $\neg y_j$  была выделена, и результирующее множество вершин было замкнуто относительно достижимости.

Упражнение 3.1 Формально обосновать корректность описанного выше сведения.

Упражнение 3.2 Доказать, что лемма 3.1 будет выполнена и для конденсации исходного графа.

На первый взгляд, ничего существенного мы пока не добились, так как размер конденсации может иметь тот же порядок, что и размер исходного графа. Есть, однако, у конденсации одно важное преимущество — это ациклический граф. Построим его топологическое упорядочение и попробуем что-нибудь сделать на его базе.

Рассмотрим некоторые две вершины конденсации:  $a$  и  $\neg a$ . Пусть для определенности вершина  $a$  находится в топологическом упорядочении раньше вершины  $\neg a$ . В общем случае, выделение вершины  $a$  не кажется разумным, так как вполне может оказаться, что в графе есть путь из  $a$  в  $\neg a$ , и тогда выделение вершины  $a$  повлечет за собой выделение  $\neg a$  и противоречие. Пути же из  $\neg a$  в  $a$  в графе точно нет, так как  $\neg a$  находится позже, чем  $a$ , в топологическом упорядочении, поэтому выделение вершины  $\neg a$  представляется более разумным шагом.

Попробуем построить алгоритм, работающий на основании этой идеи:

Построить конденсацию графа и ее топологическое упорядочение.  
**Цикл** по компонентам  $C$  конденсации в обратном топологическом порядке:

**Если** для некоторого  $i$  компонента  $C$  содержит обе вершины  $x_i$  и  $\neg x_i$ :  
 Решения не существует.

Рассмотрим любое  $i$  такое, что  $x_i$  или  $\neg x_i$  принадлежит  $C$ .

**Если** ни одна из вершин  $x_i$  и  $\neg x_i$  не выделена в исходном графе:

Выделить компоненту  $C$  в конденсации.

Выделить все вершины компоненты  $C$  в исходном графе.

Обратите внимание, что приведенный алгоритм фактически для каждого  $j$  выделяет ту из компонент  $y_j$  и  $\neg y_j$ , которая следует позже в топологическом упорядочении, но делает это без фактического введения переменных  $y_j$  (что упрощает реализацию).

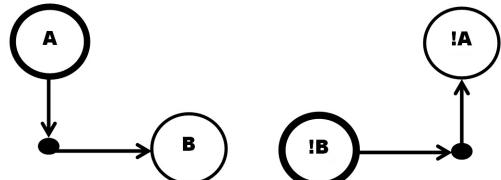
**Лемма 3.4** Выделенное алгоритмом выше множество вершин в конденсации исходного графа замкнуто относительно достижимости.

### Доказательство.

Пусть, от противного, построенное множество не замкнуто относительно достижимости. Это значит, что существуют вершины  $a$  и  $b$  такие, что вершина  $a$  выделена, вершина  $b$  — нет, и существует путь из  $a$  в  $b$ .

Заметим, что  $b$  не может быть отрицанием  $a$ , поскольку в этом случае  $\neg a$  будет стоять позже  $a$  в топологическом упорядочении, поэтому алгоритм выделит  $\neg a$ , а не  $a$ . Поэтому вершины  $a$  и  $b$  соответствуют разным переменным.

Раз вершина  $b$  не выделена, то выделена вершина  $\neg b$ , следовательно  $\neg b$  стоит позже  $b$  в топологическом упорядочении. Из наличия пути из  $a$  в  $b$  следует наличие пути из  $\neg b$  в  $\neg a$ . Таким образом,  $b$  находится позже  $a$  в топологическом порядке (из-за наличия пути из  $a$  в  $b$ ),  $\neg b$  находится позже  $b$  и  $\neg a$  находится позже  $\neg b$  (из-за наличия пути из  $\neg b$  в  $\neg a$ ). Отсюда получаем, что  $\neg a$  находится позже  $a$  в топологическом порядке. Но тогда алгоритм выделил бы вершину  $\neg a$ , а не  $a$ . Противоречие.



Из леммы 3.4 следует, что описанный выше алгоритм корректно решает задачу 2-SAT. Нетрудно видеть, что он может быть реализован так, что время работы будет линейным  $O(n + m)$ .

### Часть 3.5 Примеры задач.

В общем случае, если в задаче требуется принять ряд решений, для каждого из которых есть ровно два варианта, так, чтобы удовлетворились

некоторые ограничения, это означает сводимость задачи к SAT. Для каждого решения вводится булева переменная, одно значение которой соответствует первому варианту принятия решения, а второе значение — второму варианту. Дальше вводится функция, истинная тогда и только тогда, когда все ограничения удовлетворены и решается задача SAT.

Так как SAT — NP-полная задача, от описанного подхода мало толку. Можно показать, что если каждое из ограничений затрагивает не более  $k$  решений (переменных), то задача всегда может быть сведена к  $k$ -SAT. Нас, естественно, интересует случай  $k = 2$ . Покажем, как описать произвольное ограничение, затрагивающее только две переменные, в виде формулы в 2-CNF. Это достаточно просто. Пусть есть две переменные  $A$  и  $B$ . Ограничение запрещает некоторые комбинации значений этих переменных и разрешает все остальные комбинации. Заметим, что любую конкретную комбинацию значений можно запретить при помощи одной элементарной дизъюнкции. Более точно,  $A \parallel B$  запрещает комбинацию  $A = 0, B = 0$ ,  $A \parallel !B$  запрещает  $A = 0, B = 1$ ,  $!A \parallel B$  запрещает  $A = 1, B = 0$  и  $!A \parallel !B$  запрещает  $A = 1, B = 1$ . Чтобы описать ограничение в виде 2-CNF, достаточно выписать элементарную дизъюнкцию для каждой запрещенной комбинации значений и рассмотреть конъюнкцию всех выписанных дизъюнкций.

Замечание. Если ограничения затрагивают более 2-х переменных, это еще не значит, что задача не сводима к 2-SAT. Например, может оказаться возможным сгруппировать некоторые переменные в одну, или разбить некоторые ограничения на комбинацию более простых ограничений.

Задача 3.1 Задача Н Финала ACM ICPC 2009 (оригинал условия: <http://cm.baylor.edu/ICPCWiki/attach/Problem%20Resources/2009WorldFinalProblemSet.pdf>).

Решение. Условие достаточно явно намекает на SAT. Нужно принимать решения по биллям, по каждому биллю — два варианта решения (принять или отклонить). Поэтому для каждого билля вводим переменную, значение True которой соответствует принятию билля, а значение False — его отклонению.

Какие есть ограничения на переменные — для каждого министра нужно удовлетворить строго более половины высказанных им мнений. Так как министр может высказать до 4-х мнений, на первый взгляд ограничение затрагивает до 4-х переменных, что зарождает некоторое сомнение о сводимости задачи к 2-SAT. Не стоит, однако, делать поспешных выводов (кроме того, если вы еще это не сделали, стоит прочитать замечание, расположеннное непосредственно перед этой задачей).

Все же, очевидно, ограничение  $k \leq 4$  было введено в задачу не зря, видимо жюри не умеет решать задачу для  $k > 4$ . Попробуем понять, что

следует из  $k \leq 4$ . Если  $k = 1$  или  $k = 2$ , то все мнения данного министра необходимо удовлетворить. Если же  $k = 3$  или  $k = 4$ , то необходимо отклонить не более 1-го мнения. Эквивалентный способ сформулировать то же самое: из любых двух мнений данного министра, хотя бы одно должно быть удовлетворено. В итоге мы разбили ограничение, которое, казалось бы, затрагивало 4 переменные, на совокупность ограничений, затрагивающих 2 переменные, тем самым сведя задачу к 2-SAT.

Кроме проверки, можно ли удовлетворить более половины мнений каждого министра, требуется еще выделить билли, результат голосований по которым определяется однозначно. Вспомним алгоритм из части 3.3 и определение противоречивой вершины. Пусть  $i$ -му биллю соответствует переменная  $x_i$ . Если ровно одна из вершин  $x_i$  и  $\neg x_i$  непротиворечива, то результат голосования по  $i$ -му биллю определяется однозначно. Если же обе эти вершины непротиворечивы, то возможно как принятие, так и отклонение билля (мы можем начать алгоритм из части 3.3 с переменной  $x_i$  и принять по ней любое из двух возможных решений).

Для проверки всех вершин на противоречивость придется запустить поиск в глубину из каждой вершины. С учетом того, что в графе 2B вершин и  $O(M)$  ребер, общая сложность построенного решения есть  $O(BM)$ .

Задача 3.2 Задача В из Petr Mitrichev Contest 1 (оригинал условия: <http://acm.sgu.ru/problem.php?contest=0&problem=307>).

Решение. В отличие от предыдущей задачи, после прочтения этой совершенно не очевидно, что эта задача сводится к 2-SAT, скорее можно подумать, что это задача на решение системы линейных уравнений. Тем не менее, задача все же сводится к 2-SAT, хоть и посредством некоторой манипуляции с имеющимися уравнениями.

Заметим, что если значения  $A_{ij}$  зафиксированы в первой строке и в первом столбце, то остальные значения определяются однозначно. В самом деле, для их вычисления можно применить следующий псевдокод:

```
Для i от 2 до N:  
Для j от 2 до W:  
     $A_{ij} := B_{ij} - A_{i-1,j} - A_{i,j-1} - A_{i-1,j-1}$ 
```

Основная проблема состоит в том, что при неудачном выборе значений в первой строке и первом столбце некоторые из оставшихся значений  $A_{ij}$  могут получиться менее 0 или более 1.

В принципе, из псевдокода выше понятно, что  $A_{ij}$  является линейной комбинацией некоторых значений матрицы А из первого столбца и первой строки, к которой, возможно, прибавлена некоторая константа. Попробуем

понять, какие именно элементы входят в эту линейную комбинацию. Для начала попробуем произвести вручную вычисления для матрицы 5x5 (символом \* обозначается возможная константа, реальное значение которой за ненадобностью опущено):

$A_{11}$	$A_{12}$	$A_{13}$	$A_{14}$	$A_{15}$
$A_{21}$	$* - A_{11} - A_{12} - A_{21}$	$* + A_{11} - A_{13} + A_{21}$	$* - A_{11} - A_{14} - A_{21}$	$* + A_{11} - A_{15} + A_{21}$
$A_{31}$	$* + A_{11} + A_{12} - A_{31}$	$* - A_{11} + A_{13} + A_{31}$	$* + A_{11} + A_{14} - A_{31}$	$* - A_{11} + A_{15} + A_{31}$
$A_{41}$	$* - A_{11} - A_{12} - A_{41}$	$* + A_{11} - A_{13} + A_{41}$	$* - A_{11} - A_{14} - A_{41}$	$* + A_{11} - A_{15} + A_{41}$
$A_{51}$	$* + A_{11} + A_{12} - A_{51}$	$* - A_{11} + A_{13} + A_{51}$	$* + A_{11} + A_{14} - A_{51}$	$* - A_{11} + A_{15} + A_{51}$

После внимательного, или даже не очень внимательного рассмотрения данной таблицы, несложно заметить, что  $A_{ij}$  представляется в виде  $*_{ij} + (-1)^{i+j-1}A_{1,1} + (-1)^{j-1}A_{i,1} + (-1)^{i-1}A_{1,j}$ . При желании, эту формулу несложно доказать, например, по индукции, хотя в случае реального контекста, пожалуй, это все-таки было бы тратой времени.

Переберем два варианта для  $A_{1,1}$ , после чего это значение будем считать фиксированным. Тогда каждое  $A_{ij}$  будет зависеть только от  $A_{i,1}$  и  $A_{1,j}$ . Совершенно неожиданно получили лежащее на поверхности сведение к 2-SAT. Для каждого  $A_{i,1}$  и  $A_{1,j}$  введем булеву переменную. Для каждого  $A_{ij}$  наложим ограничения на переменные, соответствующие  $A_{i,1}$  и  $A_{1,j}$ , обеспечивающие выполнение  $A_{ij} = 0$  или  $A_{ij} = 1$ . Осталось всего лишь решить получившуюся задачу 2-SAT.

Так как количество переменных есть  $O(N)$ , а количество ограничений —  $O(N^2)$ , то исходная задача разрешима за (линейное от размера ввода) время  $O(N^2)$ . Однако с учетом ограничения  $N \leq 300$  представляется более целесообразным применение алгоритма из части 3.3. Время работы тогда составит  $O(N^3)$ .

Задача 3.3 (для самостоятельного решения). Задача Е со SWERC'2010 (оригинал условия: <http://147.96.80.25/swerc/SWERC-set.pdf>).

## Задачи и разборы

### Задача А. Anniversary

Имя входного файла: a.in  
Имя выходного файла: a.out  
Ограничение по времени: 1 с  
Ограничение по памяти: 64 Мб

В 2134 году, когда население Земли достигло  $10^{18}$  человек, было решено отметить это событие следующим образом. На огромной территории, спе-

циально отведенной под празднование, было размещено  $N$  миниатюрных лампочек (линейные размеры одной лампочки составляли менее 0.1 миллиметра), некоторым образом пронумерованных последовательными числами от 1 до  $N$ .

Изначально все лампочки были выключены. Далее было произведено ровно  $10^{18}$  шагов — по одному в честь каждого жителя Земли. На  $i$ -ом шаге одновременно изменялось состояние всех лампочек, номера которых делятся нацело на  $i$ . Изменение состояния означает, что если лампочка была выключена, то она становится включенной, и наоборот, если лампочка была включена, то она становится выключенной. Промежуток времени между последовательными шагами составил 1 пикосекунду, таким образом, все празднование заняло порядка полутора недель.

Вообще, для сторонних наблюдателей все это выглядело как беспорядочное бессмысленное мерцание, тем не менее, все были в восторге — настолько очевиден был грандиозный масштаб этого замечательного мероприятия!

Но вот все закончилось. Смотреть стало не на что и, как показал более трезвый анализ, смотреть, собственно, и было не на что.

Тем временем, после выполнения шага  $10^{18}$ , некоторые лампочки остались во включенном состоянии. Пока жители Земли отходят от шока, размышляя о том, зачем им понадобилось подобное празднование и как теперь покрыть баснословные затраты на всю эту впечатляющую ерунду, вам предлагается посчитать количество лампочек, которые все еще горят и расходуют драгоценную электроэнергию.

## **Ограничения**

Число  $N$  — целое.

$$1 \leq N \leq 2^{63} - 1.$$

## **Формат входного файла**

Первая строка входного файла содержит целое число  $N$ .

## **Формат выходного файла**

Выведите единственное целое число — количество лампочек, которые остались во включенном состоянии по завершении шага с номером  $10^{18}$ .

## **Пример**

<b>a.in</b>	<b>a.out</b>
1	1
9	3
100	10

## Разбор задачи A. Anniversary

Начнём с рассмотрения случая  $N \leq 10^{18}$ . Очевидно, что для любой лампочки  $i$  её состояние изменилось столько раз, сколько у числа  $i$  делителей. Так как изначально все лампочки были выключены, количество изменений состояния, а соответственно и делителей у числа  $i$ , должно быть нечетным. А число делителей нечетно только у полных квадратов. Поэтому ответ задачи равен целой части  $\sqrt{N}$ .

Когда  $N$  превосходит  $10^{18}$ , состояния последних  $(N - 10^{18})$  лампочек меняются уже не столько раз, сколько делителей имеют их номера: делители, превосходящие  $10^{18}$ , уже нельзя учитывать. Рассмотрим каждый из интервалов  $(10^{18}, 2 \cdot 10^{18}]$ ,  $(2 \cdot 10^{18}, 3 \cdot 10^{18}]$ ,  $\dots$ ,  $(9 \cdot 10^{18}, 10 \cdot 10^{18}]$  (те, в которых правая граница превосходит  $N$ , урежем справа до  $N$ , то есть некоторые могут получиться пустыми).

В интервале  $(10^{18}, 2 \cdot 10^{18}]$  у каждого числа есть ровно один делитель, превосходящий  $N$  — само это число. Соответственно, состояние лампочек с номерами из этого интервала, для которых номер является полным квадратом, изменилось уже четное количество раз, а всех остальных — нечетное.

В интервале  $(2 \cdot 10^{18}, 3 \cdot 10^{18}]$  каждое число “потеряло” один делитель, равный себе, а каждое четное число  $2k$  в придачу потеряло ещё один делитель  $k$ . То есть число зажженных лампочек из этого интервала равно сумме количества нечетных чисел, не являющихся полными квадратами, и четных чисел, являющихся полными квадратами.

В интервале  $(3 \cdot 10^{18}, 4 \cdot 10^{18}]$  числа, кратные трём, также потеряли делитель. Заметьте, что между числами вида  $2k$  и  $3k$  есть общие элементы, то есть теперь нам надо рассматривать отдельно числа вида  $\{2k\} - \{6k\}$ , отдельно  $\{3k\} - \{6k\}$ , отдельно  $\{6k\}$  и отдельно все остальные, и в каждом из этих множеств уметь находить количество полных квадратов и других чисел. Обобщая это с учетом следующих интервалов, мы получаем следующий метод.

В каждом интервале  $(L, R]$ , где  $L = i \cdot 10^{18}$ ,  $R = \text{MIN}(N, (i + 1) \cdot 10^{18})$ , поделим числа на группы в зависимости от их остатка при делении на  $M = \text{LCM}(1, 2, \dots, i)$ , где  $\text{LCM}$  обозначает наименьшее общее кратное. Заметьте, что при этом для любой конкретной группы однозначно определяется остаток при делении на каждое из чисел  $1, 2, \dots, i$ . Найдём количество полных квадратов и остальных чисел в каждой из таких групп. Для группы с остатком  $r$  обозначим их соответственно  $S1[r]$  и  $S0[r]$ . Для каждой из групп в ответ должно в итоге пойти  $S1[r]$ . Теперь для каждого  $j \leq i$  возьмём все группы, числа в которых делятся на  $j$ , и поменяем значениями соответствующие  $S0[r]$  и  $S1[r]$ , так как мы нашли очередной делитель,

который числа из этих групп потеряли.

Собрав для каждого интервала и для каждой группы числа  $S1[r]$  и прибавив ко всему этому  $10^9$  (количество включенных лампочек с номерами, не превосходящими  $10^{18}$ ), мы получим ответ задачи.

Один момент, который нетривиален в этом методе — каким образом считать изначальные количества  $S1[r]$  и  $S0[r]$ .  $S0[r]$  можно посчитать как общее количество чисел в заданном интервале, дающих остаток  $r$  при делении на  $M$ , минус  $S1[r]$ . А для подсчёта  $S1[r]$  нужно определить все числа  $q$ , квадрат которых по модулю  $M$  равен  $r$ , и для каждого из них посчитать количество чисел в интервале  $(\sqrt{L}, \sqrt{R})$ , которые дают остаток  $q$  при делении на  $M$ .

## Задача В. Trade

Имя входного файла:	b.in
Имя выходного файла:	b.out
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Торговля — штука тонкая. Успешный торгаш должен уметь не только вовремя продавать нужные товары и заправски зазывать клиента, но и досконально знать ситуацию на рынке. Среди всего прочего важно знать, кто из других торговцев торгует друг с другом, а кто нет. Бывает, что торговцы не общаются напрямую, но их товар всё равно попадает друг к другу через других торговцев. Например, если торговцы A и B торгуются напрямую, и торговцы B и C торгуются напрямую, то товары A и C будут попадать друг к другу через торговца B. Очень важно понятие *неразлучных пар* — это такие пары торговцев, которые торгуют напрямую, и нет других торговцев, через которых эта пара смогла бы обмениваться товаром не напрямую.

Манао хочет стать успешным торгашем. Мы не знаем, какими необходимыми качествами он обладает, но знания о ситуации на рынке ему определённо не хватает. Всё, что ему известно — что на рынке есть N торговцев, M их пар торгуют напрямую, а K из этих пар являются к тому же неразлучными. А требуется ему более конкретная информация в стиле “A торгует с B, C торгует с D, X торгует с A”. Это всё, конечно, не всегда однозначно, но на данный момент любой подходящий под описание расклад сойдёт.

Вам дают T сценариев, в каждом из которых свои значения N, M и K. Для каждого определите, соответствует ли он какой-либо торговой сети и если да, то выведите её описание. Торговцев пронумеруем в каком-либо

порядке от 1 до N, а описанием торговой сети назовём все различные пары торговцев, торгующих напрямую.

## Ограничения

$$2 \leq N \leq 100.$$

$$0 \leq K \leq M \leq N \cdot (N - 1)/2.$$

Количество сценариев в одном входном файле не превышает 100.

Сумма M по всем сценариям в одном входном файле не превышает 50,000.

## Формат входного файла

Первая строка содержит количество сценариев T. Далее следует T строк, каждая из которых содержит три числа N, M, K.

## Формат выходного файла

Для каждого из T сценариев выведите “NO SOLUTION” (без кавычек), если соответствующей торговой сети не существует. В противном случае выведите “TRADE MARKET FOUND”, а далее M строк. Каждая из строк должна содержать пару чисел, разделённых пробелом — номера торговцев, обменивающихся товаром напрямую.

## Пример

b.in	b.out
3	TRADE MARKET FOUND
4 3 0	1 2
4 2 0	2 3
5 5 2	3 1
	NO SOLUTION
	TRADE MARKET FOUND
	1 2
	2 3
	3 1
	1 4
	1 5

## Разбор задачи B. Trade

Представим искомую торговую сеть в виде графа, где торговцы соответствуют вершинам, а между парами, торгующими напрямую, в графе проведено ребро. Тогда неразлучным парам соответствуют ребра, не находящиеся в циклах — их также называют мостами.

Рассмотрим критерии, при которых искомый граф не строится:

- 1) Наибольшее количество ребер, не образующих цикл в графе из  $N$  вершин, равно  $(N-1)$ . Поэтому при  $K > N-1$  граф построить не удастся.
- 2) Если  $M = K+1$ , то нас просят сделать  $K$  мостов и одно ребро, находящееся в цикле. Цикл из одного ребра можно создать лишь соединив вершину саму с собой, что условие запрещает. Аналогично, при  $M = K+2$  нам нужно создать цикл из двух ребер, для чего нам бы потребовалось провести некоторое ребро дважды — а это в данной постановке задачи также не позволено.
- 3) Из-за того, что некоторые ребра должны быть мостами, мы в общем случае уже не можем построить граф с  $\frac{N \cdot (N-1)}{2}$  ребрами. Покажем, что в графе из  $N$  вершин, содержащем  $K$  мостов, может быть не более  $\frac{(N-K) \cdot (N-K-1)}{2}$  ребер в циклах. Обозначим  $F(x) = \frac{x \cdot (x-1)}{2}$ . Уничтожим в графе все мосты, вследствие чего он распадётся на по крайней мере  $(K+1)$  связную компоненту. Для начала допустим, что их ровно  $K+1$ . Количество вершин в компоненте  $i$  обозначим  $x(i)$ . У нас возникает задача максимизации  $F(x(1)) + F(x(2)) + \dots + F(x(K+1))$  при условии  $x(1) + x(2) + \dots + x(K+1) = N$ . Легко доказать, что  $F(a) + F(b) \leq F(a-1) + F(b+1)$  для любых двух чисел  $a$  и  $b$ . Следовательно, максимум функции будет достигнут при размерах компонентов  $\{(N-K), 1, 1, \dots\}$ , и он будет равен  $F(N - K) + K \cdot F(1) = \frac{(N-K) \cdot (N-K-1)}{2}$ . Из этого же утверждения следует, что в изначально несвязном графе не может быть большее количество вершин.

Во всех остальных случаях граф строится. Приведем пример того, как это можно сделать. Возьмём простой цикл из  $S$  вершин, где  $S$  — наименьшее число, для которого  $\frac{S \cdot (S-1)}{2} \leq M - K$  (в случае  $M = K$  возьмём просто одну вершину). Далее присоединим к этому циклу  $K$  других вершин, каждую одним ребром. Все такие ребра соответственно будут мостами. Оставшееся количество ребер восполним, проводя любые ребра внутри созданного цикла.

## Задача C. Grid

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

На плоскости задана бесконечная сетка из вертикальных и горизонтальных прямых. Эти прямые заданы уравнениями  $\{X = i \cdot K\}$  и  $\{Y = j \cdot K\}$  для любых целых чисел  $i$  и  $j$ . Также задано  $N$  точек. Для конкретной точки мы говорим, что она *принадлежит сетке*, если эта точка принадлежит хотя бы одной из прямых, образующих сетку.

Есть возможность двигать сетку параллельно осям координат. Перенос сетки на вектор  $(dx, dy)$  означает, что вместо каждой из прямых  $\{X = i \cdot K\}$  возникает прямая  $X = i \cdot K + dy$ , а вместо каждой из прямых  $\{Y = j \cdot K\}$  возникает прямая  $\{Y = j \cdot K + dx\}$ . Найдите вектор переноса сетки, в результате применения которого ей будет принадлежать наибольшее количество заданных точек.

### Ограничения

$$1 \leq N \leq 100,000 (10^5).$$
$$2 \leq K \leq 1,000,000,000 (10^9).$$

Координаты всех точек целые и по абсолютной величине не превосходят  $10^9$ .

Никакие две точки не совпадают.

### Формат входного файла

Первая строка содержит два целых числа  $N$  и  $K$ .  $i$ -ая из следующих  $N$  строк содержит пару чисел  $X_i, Y_i$  — координаты  $i$ -ой точки.

### Формат выходного файла

Выведите максимальное возможное количество одновременно принадлежащих сетке точек из заданного множества.

## Пример

<b>c.in</b>	<b>c.out</b>
6 2	5
2 0	
0 1	
2 2	
3 3	
3 2	
4 1	

## Пояснение

Один из возможных векторов переноса сетки в данном примере —  $(1,0)$ .

## Разбор задачи C. Grid

В первую очередь заметим, что при переносе сетки на вектор  $(dx + K, dy)$  мы, вследствие её бесконечности, получаем то же самое, что при передвижении её на вектор  $(dx, dy)$ . Аналогично и при передвижении в ортогональном направлении. Поэтому каждую из входных точек  $(x, y)$  можно с самого начала заменить на точку  $(x^*, y^*)$  такую, что  $0 \leq x^*$ ,  $y^* < K$ , числа  $x$  и  $x^*$  дают один и тот же остаток при делении на  $K$ , и числа  $y$  и  $y^*$  также имеют одинаковый остаток при делении на  $K$ . Некоторые точки могут возникнуть дважды, и все повторения также нужно учитывать.

Обозначим через  $cnt\{(x, y), P\}$  количество точек в нашем уже мульти множестве, удовлетворяющих условию  $P$ . Теперь задачу можно поставить так: надо найти такие  $dx$  и  $dy$  ( $0 \leq dx, dy < K$ ), что величина  $cnt\{(x, y), x = dx\} + cnt\{(x, y), y = dy\} - cnt\{(x, y), x = dx, y = dy\}$  максимальна.

Подсчитаем для каждого  $dx$   $cnt\{(x, y), x = dx\}$  и сохраним эти значения так, чтобы впоследствии быстро к ним обращаться. Также нам понадобится структура, с помощью которой мы быстро сможем определить, сколько раз встречается некоторая точка в нашем мульти множестве. Для обеих этих задач сгодится бинарное дерево поиска, которое имплементировано в стандартных библиотеках Java и C++ как `Map`. Также отсортируем пары  $(dy, cnt\{(x,y), y=dy\})$  в порядке невозрастания второй величины.

После всех этих приготовлений пройдём для каждого  $dx$  по отсортированному списку горизонталей  $dy$  до тех пор, пока не окажется, что текущая точка  $(dx, dy)$  не нашлась в нашем мульти множестве. Для всех пройденных  $dy$  проверим, не улучшает ли текущий ответ сумма

$cnt\{(x, y), x = dx\} + cnt\{(x, y), y = dy\} - cnt\{(x, y), x = dx, y = dy\}$ . Дальше по этому списку идти смысла нет, так как  $cnt\{(x, y), y = dy\}$  упорядочены по убыванию, а  $cnt\{(x, y), x = dx, y = dy\}$  меньше нуля стать не может — то есть лучшего варианта для этого  $dx$  уже не будет. Когда мы рассмотрим каждый различный  $dx$ , у нас будет найден ответ задачи.

Рассмотрим теперь сложность приведенного алгоритма. Все “подготовительные работы”, описанные в третьем абзаце, делаются за время  $O(N \cdot \log N)$ . Непосредственно алгоритм для конкретного  $dx$  может пройти по всему списку горизонталей, но для всех  $dx$  в сумме он рассмотрит не более  $N$  горизонталей. Это понятно оттуда, что каждая точка принадлежит ровно одной горизонтали и вертикали. При рассмотрении каждой горизонтали нам требуется найти значение  $cnt\{(x, y), x = dx, y = dy\}$ , хранящееся в дереве поиска и соответственно требующее  $O(\log N)$  операций. Итого эта часть алгоритма также требует  $O(N \cdot \log N)$  операций, так что итоговая сложность  $O(N \cdot \log N)$ .

## Задача D. Drawing

Имя входного файла:	d.in
Имя выходного файла:	d.out
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Маленький Вася очень любит рисовать.

Сегодня он взял таблицу из  $N$  строк и  $M$  столбцов и решил раскрасить ее клетки в разные цвета. Чтобы результат раскраски получился менее предсказуемым, он решил произвести ее следующим образом:

- Для каждого  $i$  от 1 до  $N$  с вероятностью  $R_i\%$  Вася полностью раскрашивает все клетки  $i$ -й строки в красный цвет.
- Для каждого  $j$  от 1 до  $M$  с вероятностью  $C_j\%$  Вася полностью раскрашивает все клетки  $j$ -го столбца в синий цвет.

Если клетка раскрашивается и в красный, и в синий цвета, то в результате смешения цветов она оказывается зеленой.

Изначально ни одна из клеток таблицы раскрашена не была, а по окончании применения процедуры, описанной выше,  $G$  клеток оказались закрашенными в зеленый цвет. Определите математическое ожидание общего количества клеток, раскрашенных Васей в какой-либо цвет (красный, синий или зеленый).

## Ограничения

Все числа во вводе целые.

$$1 \leq N, M \leq 100.$$

$$0 < R_i, C_j < 100.$$

$$0 \leq G \leq N \cdot M.$$

Существует хотя бы один способ раскрасить некоторые строки в красный цвет, а некоторые столбцы — в синий цвет так, что ровно  $G$  клеток окажутся закрашенными в зеленый цвет.

## Формат входного файла

Первая строка входного файла содержит три числа  $N$ ,  $M$  и  $G$ , разделенные одиночными пробелами.

Во второй строке записаны числа  $R_1, R_2, \dots, R_N$ , разделенные одиночными пробелами.

В третьей строке записаны числа  $C_1, C_2, \dots, C_M$ , разделенные одиночными пробелами.

## Формат выходного файла

Выведите искомое математическое ожидание. Абсолютная погрешность выведенного ответа не должна превышать  $10^{-5}$ .

## Пример

<b>d.in</b>	<b>d.out</b>
2 2 1 50 50 50 50	3.0
2 2 0 50 50 50 50	2.2857142857142856
2 3 2 3 2 3 2 3	5.313107297058556

## Разбор задачи D. Drawing

Допустим, мы знаем вероятности  $pR[i]$  и  $pC[j]$ , где  $pR[i]$  — вероятность того, что было закрашено  $i$  строк из  $N$ , а  $pC[j]$  — вероятность того, что было закрашено  $j$  столбцов из  $M$ . Какие именно строки и столбцы при этом закрашены, значения не имеет. Если у нас закрашено  $i$  строк

и  $j$  столбцов, тогда на их пересечении будет ровно  $i \cdot j$  клеток. А общее количество закрашенных клеток будет  $(i \cdot M + j \cdot N - i \cdot j)$ .

Нам интересны только те случаи, когда  $i \cdot j = G$ . Согласно формуле Байеса, вероятность каждого конкретного такого случая равна

$$\frac{pR[i] \cdot pC[j]}{\sum_i \sum_{\substack{j \\ i \cdot j = G}} pR[i] \cdot pC[j]}.$$

Таким образом, зная значения  $pR[i]$  и  $pC[j]$ , мы можем перебрать все возможные пары  $(i, j)$ , посчитать суммы  $pR[i] \cdot pC[j] \cdot (i \cdot M + j \cdot N - i \cdot j)$  и  $pR[i] \cdot pC[j]$  и вывести их отношение.

Для вычисления  $pR[i]$  и  $pC[j]$  применим принцип динамического программирования. Конкретно, обозначим через  $probR[i][j]$  вероятность того, что из первых  $i$  строк  $j$  будут закрашены. При этом  $probR[0][0] = 1.0$ ,  $probR[0][j] = 0.0$  для всех  $j > 0$ . Выполняется следующая рекуррентная формула:

$$probR[i][j] = probR[i-1][j] \cdot \frac{(100 - R[i])}{100} + probR[i-1][j-1] \cdot \frac{R[i]}{100}$$

Числа  $pR[i]$  являются значениями  $probR[N][i]$  по определению. Аналогичным образом можно вычислить и  $pC[j]$ .

## Задача E. Empire

Имя входного файла:	<code>e.in</code>
Имя выходного файла:	<code>e.out</code>
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Империя включает в себя  $N$  планет. Пронумеруем эти планеты числами от 1 до  $N$ . Планета с номером 1 — столица Империи, где находится резиденция Императора и подготавливаются войска. На различных планетах империи часто вспыхивают восстания, которые приходится подавлять военной силой и немедленно.

Для того, чтобы войска могли быстро передвигаться, на некоторых планетах установлены односторонние телепорты. Всего телепортов  $M$  штук. С помощью  $i$ -ого телепорта можно в мгновение ока попасть с планеты  $A_i$  на планету  $B_i$  (но не наоборот). Таким образом, вовремя подавить восстание на некоторой планете  $X$  можно, если существует последовательность планет  $P_1, \dots, P_k$  ( $k \geq 2$ ) такая, что  $P_1=1$ ,  $P_k=X$ , а для каждого  $1 \leq i < k$  есть

телеport из планеты с номером  $P_i$  на планету с номером  $P_{i+1}$ . Учитывая, что после подавления восстания войска остаются на планете для поддержания порядка, об их возвращении в столицу мы можем не беспокоиться.

Проверьте, есть ли возможность при имеющихся телепортах подавить восстание на любой планете Империи. Если это так, выведите 0. В противном случае, найдите наименьшее количество телепортов, которое надо дополнительно построить, чтобы восстание на любой планете было подавляемо. Каждый телепорт может быть построен между произвольными двумя планетами.

## Ограничения

$$2 \leq N \leq 100,000 (10^5).$$

$$0 \leq M \leq 200,000 (2 \cdot 10^5).$$

$$1 \leq A_i, B_i \leq N \text{ для всех } 1 \leq i \leq M.$$

Ни на одной планете нет телепорта в саму себя.

Ни на одной планете нет двух телепортов на одну и ту же планету.

## Формат входного файла

Первая строка содержит два целых числа  $N$  и  $M$ .  $i$ -ая из следующих  $M$  строк содержит пару чисел  $A_i, B_i$ .

## Формат выходного файла

Выведите наименьшее количество телепортов, гарантирующее, что восстание на любой планете можно будет немедленно подавить.

## Пример

<b>e.in</b>	<b>e.out</b>
6 4	2
3 1	
4 6	
1 2	
4 5	

## Пояснение

Можно построить телепорт из планеты 2 на планету 4 и из планеты 5 на планету 3.

## Разбор задачи E. Empire

Переведём поставленную задачу в термины теории графов. Заданы ориентированный граф  $G$  (вершины которого соответствуют планетам, а

дуги — телепортам) и условие “любая вершина графа должна быть достижима из вершины 1”. Обозначим это условие через  $Y$ . Требуется найти наименьшее количество дуг, одновременное добавление которых в  $G$  обеспечит, чтобы он удовлетворял  $Y$ .

Чтобы просто решить эту задачу, нужно сделать несколько по отдельности несложных наблюдений:

- 1) Существует такое оптимальное множество дуг, при добавлении которых в  $G$  он будет удовлетворять  $Y$ , и началом каждой дуги в этом множестве будет вершина 1. В этом можно убедиться, рассмотрев любое оптимальное множество, содержащее дуги, исходящие не из 1. Заменив любую такую дугу  $(A, B)$  на  $(1, B)$ , мы никоим образом не изменим достижимость вершин из вершины 1.
- 2) Рассмотрим конденсацию  $C$  этого графа. Рассмотрим вершину  $V$  в графе  $C$ , которая не имеет входящих дуг. Она соответствует некоторому сильносвязному компоненту в графе  $G$ . Нам в ответе необходима дуга из вершины 1 в какую-либо из вершин этого компонента. Единственная оговорка здесь — это когда вершина 1 входит в этот компонент. В таком случае, естественно, соответствующая дуга не требуется.
- 3) Учитывая, что  $C$  является ациклическим графом, для всех остальных вершин  $C$  дополнительных дуг не требуется: у каждой из них обязательно есть предок, в которого не входит ни одной дуги, а в него мы, согласно предыдущему наблюдению, дугу проведём обязательно.

На основе сделанных наблюдений можно построить следующий алгоритм:

- 1) Находим сильносвязные компоненты в графе.
- 2) На основе найденных компонентов строим конденсацию графа.
- 3) Находим количество вершин  $T$  в конденсации, в которые не входят дуги.
- 4) Проверяем, входит ли вершина 1 изначального графа в какой-либо из компонентов, соответствующих таким вершинам. Если да, то ответом задачи является число  $(T - 1)$ , в противном случае это число  $T$ .

## Задача F. Reform

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

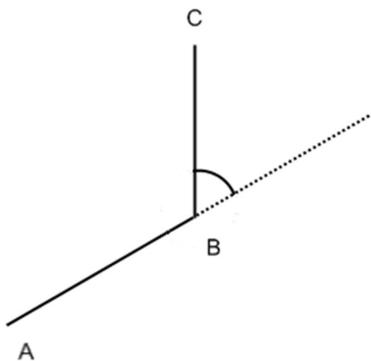
На одной замечательной плоскости с декартовой системой координат ис- покон веков была расположена очень консервативная страна. В этой стране испокон веков было  $N$  городов, пронумерованных последовательными целыми числами от 1 до  $N$ . По сравнению с бесконечностью плоскости города, даже очень большие, очень невелики, поэтому мы будем считать их точками на плоскости. Координаты  $i$ -го города —  $(X_i, Y_i)$ . Координаты любых двух городов различны. Никакие 3 города не лежат на одной прямой линии.

Некоторые пары городов соединены двунаправленными дорогами. Каждая дорога — это отрезок прямой линии, соединяющий некоторые два города. Известно, что из каждого города выходит ровно 3 дороги. Никакая дорога не соединяет город с самим собой. Между каждой парой городов может быть не более одной дороги.

Так обстояли дела в этой стране с незапамятных времен, и никому даже не приходило в голову что-либо поменять. Но вот произошла беда — к власти пришел либеральный король! И проблемы не заставили себя ждать — незамедлительно последовал указ о реформе дорожного сообщения в стране. Было приказано убрать некоторые дороги так, чтобы в результате:

- Из каждого города выходило ровно 2 дороги.
- Угол поворота между двумя дорогами, выходящими из одного и того же города, был *строго* меньше 60 градусов.
- Никакие две дороги не пересекались нигде, кроме как в городах.

Угол поворота между двумя дорогами вычисляется следующим образом. Пусть из города В дороги идут в города А и С. Тогда угол поворота между ними равен внешнему углу при вершине В в треугольнике ABC (см. рисунок).



Реализация реформы была поручена министру транспорта. Именно ему предстоит решить, какие дороги убрать, а какие оставить. Желая угодить королю, среди всех возможных способов решения поставленной королем задачи, министр хочет выбрать такой, в котором максимальный из углов поворота между двумя дорогами из одного и того же города минимален.

## Ограничения

Все числа во входных данных целые.

$4 \leq N \leq 200$ ,  $N$  — четное.

$-100000 \leq X_i, Y_i \leq 100000$ .

Координаты никаких двух городов не совпадают.

Никакие 3 города не лежат на одной прямой линии.

Каждый город соединен дорогами ровно с 3-мя городами.

Между парой городов может быть не более одной дороги.

Никакая дорога не соединяет город с самим собой.

Любые два угла поворота (не обязательно при одном городе) в стране до реформы различаются не менее чем на  $10^{-5}$  градуса.

Любой угол поворота в стране до реформы отличается от угла в 60 градусов не менее чем на  $10^{-5}$  градуса.

## Формат входного файла

Первая строка входного файла содержит целое число  $N$ . Каждая из следующих  $N$  строк содержит 5 целых чисел. Первые два числа в  $i$ -й из этих строк — это  $X_i$  и  $Y_i$ . Следующие три числа — это номера городов, с которыми  $i$ -й город изначально соединен дорогами.

## Формат выходного файла

Если поставленные королем условия выполнить невозможно, выведите единственную строку содержащую “Minister’s life is short :(” (без крайних кавычек). Символы “”, ‘:’ и ‘(’ имеют ASCII-коды 39, 58 и 40 соответственно.

Иначе выведите способ решения задачи, который следует выбрать министру, в виде  $N$  целых чисел, разделенных пробелами. Если  $i$ -е из выведенных чисел равно  $j$ , это означает, что министру следует убрать дорогу между городами  $i$  и  $j$ .

### Пример

<b>f.in</b>	<b>f.out</b>
<pre> 4 0 0 2 3 4 41 0 1 3 4 0 42 4 2 1 43 44 2 3 1 </pre>	Minister's life is short :(
<pre> 8 0 100 2 6 8 0 201 3 1 5 102 303 2 4 8 204 305 3 5 7 306 207 2 4 6 308 109 1 5 7 210 0 6 8 4 111 0 1 3 7 </pre>	6 5 8 7 2 1 4 3

### Разбор задачи F. Reform

Для начала научимся просто удовлетворять требования короля, без дополнительной максимизации минимального угла поворота. Эта задача может быть сведена к 2-SAT.

Ключевое наблюдение здесь следующее. На первый взгляд, в каждом городе есть три варианта выбора дороги для удаления. Однако более внимательное рассмотрение показывает, что с учетом дополнительного ограничения, что угол поворота между оставшимися двумя дорогами должен быть строго меньше 60 градусов, на самом деле этих способов не более чем два. В самом деле, пусть из некоторого города  $A$  выходят дороги в города  $B$ ,  $C$  и  $D$ . Рассмотрим углы  $BAC$ ,  $CAD$  и  $DAB$ . Их сумма равна 360 градусам, следовательно, минимальный из них составляет не более чем  $360 / 3 = 120$  градусов. Отсюда угол поворота между двумя дорогами, образующими этот угол, составляет не менее 60 градусов.

Рассмотрим для каждого города, сколько вариантов в нем так удалить дорогу, чтобы угол между двумя оставшимися был менее 60 градусов. Если для некоторого города их 0, то, очевидно, министру не сносить головы.

Если вариант один, то его и надо выбирать. Если же варианта два, введем для этого города булеву переменную. Будем считать, что значение этой переменной TRUE соответствует одному из вариантов удаления дороги, а значение FALSE соответствует другому варианту.

Посмотрим, какие ограничения надо наложить на значения переменных, чтобы выполнить все требования короля. Таких ограничений всего лишь два типа:

- Если в городе  $A$  удаляется дорога между городами  $A$  и  $B$ , то в городе  $B$  также должна удаляться именно дорога между городами  $A$  и  $B$ .
- Если две дороги пересекаются (не в городе), то одну из них обязательно следует удалить.

Как видим, каждое из этих ограничение затрагивает только два города (во втором ограничении каждую дорогу можно приписать одному из ее концевых городов). Поэтому (см. лекцию) совокупность всех ограничений может быть описана формулой в виде 2-CNF. С учетом того, что количество переменных в ней порядка  $O(N)$ , а количество ограничений — порядка  $O(N^2)$ , мы можем проверить выполнимость этой формулы за время  $O(N^2)$ , что более чем допустимо для  $N \leq 200$ .

Вернемся к исходной задаче. В ней дополнительно требуется минимизировать максимальный угол поворота между оставшимися дорогами. Это достаточно типичная ситуация для применения двоичного поиска. Заменим исходное ограничение “угол поворота  $< 60$  градусов” на “угол поворота  $\leq X$  градусов”, где  $X$  — это некоторая переменная. Для минимизации максимального угла поворота нужно найти минимальное  $X$ , для которого все ограничения все еще удается удовлетворить. Если ограничения удается удовлетворить для  $X = X_0$ , то их можно удовлетворить и для всех  $X$ , больших  $X_0$ . С учетом этого для поиска порогового  $X$ , начиная с которого все ограничения удается удовлетворить, можно применить двоичный поиск.

Дополнительно можно заметить, что в качестве значений  $X$  имеет смысл рассматривать только имеющиеся изначально углы поворота между дорогами. Таких углов —  $O(N)$ , поэтому двоичный поиск потребует  $O(\log N)$  итераций. На каждой итерации потребуется проверять 2-CNF формулу на выполнимость, поэтому общая сложность решения —  $O(N^2 \cdot \log N)$ .

## Задача G. Constellation

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

Недавно исследователи обнаружили древний манускрипт, который описывает некоторое созвездие. Из него следует, что созвездие состоит из  $M$  звёзд, а также известны расстояния между каждой парой звёзд созвездия, если рассматривать небо как плоскость, а звёзды — как точки на этой плоскости.

На сегодняшнем небе в том полушарии, где предположительно находится описанное созвездие, видно  $N$  звёзд. Обычно созвездием считают объединение самых ярких звёзд какого-либо фрагмента неба, но за прошедшие тысячелетия яркости звёзд могли измениться, поэтому опираться на этот показатель уже невозможно. Следовательно, определять, какие звёзды сегодняшнего неба могут образовывать описанное в манускрипте созвездие, приходится лишь на основе данных расстояний.

Будем называть *возможным местоположением созвездия* список звёзд ( $I_1, I_2, \dots, I_M$ ) такой, что для каждого  $i$  и  $j$  ( $1 \leq i, j \leq M$ ) расстояние между звёздами  $I_i$  и  $I_j$  равно расстоянию между  $i$ -ой и  $j$ -ой звёздами манускрипта. Два возможных местоположения различны, если хотя бы на одной из позиций в них записаны разные звёзды.

Вам задано множество звёзд на сегодняшнем небе. Также дана матрица размерности  $M \times M$ , где элемент  $(i, j)$  обозначает квадрат расстояния между звёздами  $i$  и  $j$  созвездия. Подсчитайте количество возможных местоположений этого созвездия.

### Ограничения

$$2 \leq N \leq 30,000.$$

$$2 \leq M \leq \min(N, 20).$$

Координаты каждой звезды — целые числа, не превосходящие 10,000 по абсолютному значению. Никакие две звезды не находятся в одной точке.

Заданная матрица расстояний симметрична, её главная диагональ содержит только нули, а все остальные числа положительные и не превосходят  $10^8$ .

### Формат входного файла

Первая строка входного файла содержит число  $M$ . Следующие  $M$  строк содержат по  $M$  целых чисел каждая — матрицу расстояний. Далее запи-

сано число  $N$ , а затем следует  $N$  строк, каждая из которых содержит пару целых чисел  $X_i, Y_i$  — координаты  $i$ -ой звезды на сегодняшнем небе.

## Формат выходного файла

Выведите количество возможных местоположений созвездия из манускрипта.

## Пример

<b>g.in</b>	<b>g.out</b>
3 0 1 2 1 0 1 2 1 0 4 0 0 1 0 0 1 1 1	8

## Пояснение

Возможными местоположениями созвездия являются  $(1,2,4)$ ,  $(1,3,4)$ ,  $(2,1,3)$ ,  $(2,4,3)$ ,  $(3,1,2)$ ,  $(3,4,2)$ ,  $(4,2,1)$ ,  $(4,3,1)$ .

## Разбор задачи G. Constellation

Допустим, мы выбрали какие-либо две звезды  $s1$  и  $s2$  в наше созвездие. Без потери общности можно считать, что в манускрипте они идут под номерами 1 и 2. Матрицу манускрипта обозначим через  $D(i, j)$ . Так как третья звезда должна находиться на расстоянии  $\sqrt{D(1, 3)}$  от первой и на расстоянии  $\sqrt{D(2, 3)}$  от второй, она должна быть одной из точек пересечения двух окружностей с центрами в  $s1$  и  $s2$  и радиусами  $\sqrt{D(1, 3)}$  и  $\sqrt{D(2, 3)}$  соответственно. В худшем случае таких точек две. Тогда при выборе следующих звезд у нас будет не более одного возможного кандидата (так как три окружности с центрами не на одной прямой не могут иметь более одной общей точки). В том случае, когда при выборе третьей звезды у нас только один кандидат, все три первые точки оказываются на одной прямой, поэтому кандидатов на следующие звезды может быть два — но после одного такого выбора все остальные звезды уже определяются однозначно. Поэтому ответ задачи не может превосходить [количество упорядоченных

пар с расстоянием  $\sqrt{D(1, 2)}$  друг от друга] · 2, как в принципе и удвоенное количество упорядоченных пар с расстоянием  $\sqrt{D(i, j)}$  для любых  $i \neq j$ .

Попытаемся оценить количество пар звёзд с некоторым фиксированным расстоянием. Лучшая оценка сверху этого числа, которую можно встретить в литературе, —  $O(N^{\frac{4}{3}})$ , и это когда позволены любые координаты. В нашем случае они целые и ограничены по модулю десятью тысячами, поэтому таких пар относительно немного. В наших тестах наибольшее отношение количества пар к  $N$  чуть больше 8.8 (для  $N=30000$ ,  $P=264284$ ).

Таким образом, для каждой подходящей пары звёзд нам надо построить упомянутые выше окружности, найти их точки пересечения и рекурсивно попытаться взять третьей звездой каждую из этих точек (если они имеются в заданном множестве — небе). Для каждого из вариантов следует построить окружности с радиусами  $\sqrt{D(1, 4)}$ ,  $\sqrt{D(2, 4)}$  и  $\sqrt{D(3, 4)}$  и найти их общие точки, и так далее. Заметим, что строить можно только две окружности и для найденных точек пересечения проверять, принадлежат ли они оставшимся. Получается, что мы не более чем  $(M - 2)$  раза строим окружности и находим их пересечение, для каждого такого построения и каждой из точек пересечения проверяем  $(M - 2)$  окружности на предмет содержания этой точки и проверяем, что точка имеется в заданном множестве. Итого сложность алгоритма  $O(P \cdot M \cdot (M + C))$ , где  $P$  — количество подходящих пар, а  $C$  — сложность проверки принадлежности данной звезды заданному множеству.

Проверку принадлежности можно выполнить за  $O(\log N)$ , если построить из звезд дерево поиска. Можно уменьшить сложность проверки до  $O(1)$ , если сохранить всё небо в квадратный массив, используя каждый бит для кодирования одной точки.

Чтобы не проверять все возможные пары звёзд при поиске подходящих пар, заранее найдём все такие  $(a, b)$ , что  $a^2 + b^2 = D(1, 2)$ . Далее, выбрав некоторую точку  $(x, y)$  как первую звезду, рассмотрим все точки вида  $(x + a, y + b)$ ,  $(x + a, y - b)$ ,  $(x - a, y + b)$ ,  $(x - a, y - b)$  и проверим каждую на принадлежность заданному множеству.

## Задача Н. Presents

Имя входного файла:	<code>h.in</code>
Имя выходного файла:	<code>h.out</code>
Ограничение по времени:	3 с
Ограничение по памяти:	64 Мб

Манао решил устроить вечеринку. Он ярко украсил свою хату, тщательно подобрал меню, созвал узкий круг друзей... и вдруг осознал, что забыл

про подарки!

На своих вечеринках Манао всегда дарит гостям подарки. Более того, он всегда дарит каждому что-нибудь такое, чему тот будет рад. Сами гости тоже приносят подарки, каждый по одной штуке. Так как времени осталось немного, Манао не успеет накупить подарков, поэтому ему нужно найти способ выкрутиться из ситуации. И тогда его осеняет гениальная идея: перераспределить принесённые гостями подарки между ними! Естественно, отдать человеку принесённый им же подарок нельзя. Манао хорошо знает своих друзей и заранее может сказать, кому из них понравится принесённый кем подарок.

Но, к сожалению, может так случиться, что распределить подарки так, чтобы все друзья были рады, не получится. Поэтому Манао решил позвать вдобавок ещё некоторое количество знакомых. Манао хорошо знает и их, и поэтому может с уверенностью предсказать, кто что принесёт и чему будет рад. Обижать знакомых Манао тоже не хочет, поэтому каждый из них также должен получить подарок, которому будет рад. Но чтобы не превращать вечеринку в полный балаган, Манао пригласит наименьшее количество знакомых, которое обеспечит то, что распределить подарки между всеми гостями так, чтобы каждый был рад, получится.

У Манао  $N$  друзей и  $M$  других знакомых. Пронумеруем друзей числами от 1 до  $N$ , а знакомых числами от  $N + 1$  до  $N + M$ . Вам дана матрица, по которой можно определить, чей подарок кому понравится. Определите наименьшее количество знакомых, которых надо дополнительно позвать. Если это невозможно, выведите число -1.

## Ограничения

$$2 \leq N \leq 100.$$

$$0 \leq M \leq 100.$$

Ни одному гостю не может понравиться свой же подарок.

## Формат входного файла

Первая строка входного файла содержит два числа  $N$  и  $M$ . Каждая из следующих  $N + M$  строк содержит по  $N + M$  символов "Y" или "N".  $j$ -ый символ в  $i$ -ой из этих строк обозначает, понравится ли человеку с номером  $i$  подарок человека с номером  $j$ .

## Формат выходного файла

Выведите -1, если никакое множество знакомых не спасёт вечеринку Манао, и размер такого минимального множества в противном случае.

## Пример

<b>h.in</b>	<b>h.out</b>
2 2 NYYY NNYN YNNN NYNN	1
3 2 NNYN YNYNN NNYN NYNNY NNYNN	-1
3 2 NYN YNYNN NNYN NYNNY NNYNN	2

## Пояснение

В первом примере можно пригласить человека с номером 3, а потом отдать гостю 2 подарок, принесённый гостем 3, гостю 3 подарок, принесённый гостем 1, а гостю 1 подарок гостя 2. В третьем примере можно, пригласив обоих знакомых, обменять подарки первого и второго друзей, четвертому отдать подарок третьего, третьему — пятого, а пятому — четвертого.

## Разбор задачи H. Presents

Рассмотрим двудольный граф, состоящий из  $2 \cdot (N + M)$  вершин — две на каждого потенциального гостя Манао, причём одна из этих вершин будет в левой доле графа, а вторая в правой. Для человека с номером  $x$  обозначим через  $L(x)$  соответствующую ему вершину из левой доли и через  $R(x)$  вершину из правой. Для каждой пары гостей  $(a, b)$  проведём из  $L(a)$  в  $R(b)$  ребро с проводимостью 1, если гостю  $b$  понравится подарок, который принесёт гость  $a$ . Также для каждого из гостей  $i$ , которого приглашать необязательно, создадим ребро с проводимостью 1 из  $L(i)$  в  $R(i)$ .

Найдя максимальное паросочетание в построенном графе, мы сможем ответить на один важный вопрос: существует ли вообще некоторое множе-

ство гостей, содержащее всех друзей Манао, для которого перераспределение подарков возможно. Существует оно только тогда, когда существует паросочетание, включающее все вершины. Можно заметить соответствие между полным паросочетанием в построенном графе и некоторым множеством приглашенных гостей, включающим всех друзей. Для этого соединим все вершины  $L(i)$  и  $R(i)$  в одну вершину и рассмотрим полученный граф. Он будет состоять из непересекающихся простых циклов длины 2 или больше и одиночных вершин. Каждая одиночная вершина — это неприглашенный гость. А в каждом цикле подарки можно подарить по кругу.

Чтобы найти минимальное множество, удовлетворяющее поставленным условиям, мы поставим на ребра стоимости. Установим стоимость всех ребер вида  $L(i) \rightarrow R(i)$  равной -1, а все остальные пусть имеют стоимость 0. Теперь найдём максимальное паросочетание минимальной стоимости в таком графе. Его стоимость и будет ответом задачи.

## Задача I. ICPC

Имя входного файла:	<code>i.in</code>
Имя выходного файла:	<code>i.out</code>
Ограничение по времени:	3 с
Ограничение по памяти:	64 Мб

Вот уже много лет существует мрачный куль ICPC. По свидетельствам очевидцев, служители этого культа занимаются очень странными вещами. Они разбиваются на группы по 3 человека и поклоняются экрану, от которого исходит таинственное свечение, с силой бьют по каким-то кнопкам, сгруппированным на непонятного рода панели. Коммуникация эта, видимо, не является однодirectionalной — иногда “божество” подает какие-то знаки одобрения в ответ, и тогда раздаются дружные радостные крики служителей. У большинства нерадивых служителей, правда, ситуация эта происходит довольно редко. Чаще “божество” недовольно, что повергает поклоняющихся сначала в замешательство, а потом, нередко, и в уныние.

О природе “божества” ходят разные слухи. Некоторые утверждают, что “божество” существует во многих обличьях, каждое из которых управляет посвященным человеком (а может это и не человек вовсе, а существо более высокого порядка) — Админом. Особенную известность получил так называемый Админ с Большой Бородой. А еще с давних пор ходит слух об особой благосклонности “божества” к кефиру... Впрочем, это уже какая-то нелепица, да и от темы задачи мы отвлеклись.

А вот собственно и сама задача. Собрались как-то раз  $N$  служителей культа ICPC и решили, что традиционные методы поклонения устарели,

надоели, да и вообще какие-то не особо зловещие. Постановили они, что вместо усиленного ожесточенного стука по клавиатуре следует им обмениваться темной энергией. Был разработан план — множество из  $M$  пар  $(A, B)$ , означающих, что служитель с номером  $A$  должен передавать энергию служителю с номером  $B$  (соответственно, служитель с номером  $B$  должен получать энергию от служителя с номером  $A$ ).

Начали они действовать согласно плану, но энергия почему-то не передавалась. Как вы сами понимаете, проблема ну никак не может быть в том, что нет у служителей ICPC никакой темной энергии! Понимали это и сами служители. Спустя некоторое время осознали они, что для того, чтобы план сработал, следует им расположиться определенным образом. Начертили они магический круг и расположились по его границам.

Конечно, одного наличия круга недостаточно, требуется еще, чтобы процесс передачи энергии был ассоциирован с кругом. Общеизвестно, что передача энергии ассоциирована с кругом, если каждый служитель передает энергию непосредственно следующим за ним в круге (при обходе круга по часовой стрелке) служителям и получает энергию от непосредственно следующих перед ним в круге служителей.

Определим то же самое более формально. Пусть  $next(X)$  — номер служителя, стоящего непосредственно за служителем  $X$  по кругу (при обходе круга по часовой стрелке), а  $prev(X)$  — номер служителя, стоящего непосредственно перед служителем  $X$ . Определим для  $i > 1$  величины  $next^i(X) = next(next^{i-1}(X))$  и  $prev^i(X) = prev(prev^{i-1}(X))$  (будем также считать, что  $next^1(X) = next(X)$  и  $prev^1(X) = prev(X)$ ). Передача энергии ассоциирована с магическим кругом, если для каждого служителя  $X$  выполнены следующие условия:

- он передает энергию служителям  $next^i(X)$ ,  $1 \leq i \leq A$ , где  $A$  — общее количество служителей, которым он передает энергию;
- он получает энергию от служителей  $prev^i(X)$ ,  $1 \leq i \leq B$ , где  $B$  — общее количество служителей, от которых он получает энергию.

Никогда еще цель не была так близка, однако расположиться по кругу таким образом, чтобы достичь ассоциированной передачи энергии, у собравшихся служителей ICPC не получилось. Как братья по культуре ICPC, вы, конечно же, приложите все усилия, чтобы помочь им!

## Ограничения

Все числа во входном файле целые.

$N \geq 3$ .

Сумма значений  $N$  по всем тестам в одном файле не превосходит 100 000.

Сумма значений  $M$  по всем тестам в одном файле не превосходит 200 000.

$1 \leq A, B \leq N$ .

$A \neq B$ .

В рамках одного теста каждая пара  $(A, B)$  присутствует не более одного раза.

В рамках одного теста не могут одновременно присутствовать обе пары  $(A, B)$  и  $(B, A)$ .

## **Формат входного файла**

Входной файл содержит несколько тестов. В первой строке файла записано их количество  $T$ , далее следует описание  $T$  тестов.

Описание каждого теста начинается со строки, содержащей числа  $N$  и  $M$ . Далее следует  $M$  строк, описывающих план передачи энергии. Каждая из этих строк содержит два целых числа —  $A$  и  $B$ . Служители пронумерованы числами от 1 до  $N$ .

## **Формат выходного файла**

Для каждого теста во входном файле выведите строку, содержащую перестановку чисел от 1 до  $N$  — порядок, в котором следует стать служителям, чтобы достичь передачи энергии, ассоциированной с магическим кругом. Выденный порядок должен соответствовать обходу круга по часовой стрелке. Если существует несколько решений, выведите любое из них. Если решения не существует, выведите “Epic fail” (без кавычек).

## Пример

<b>i.in</b>	<b>i.out</b>
3	1 6 2 4 5 3
6 10	1 2 3
1 6	Epic fail
2 3	
2 4	
2 5	
3 1	
3 6	
4 3	
4 5	
5 3	
6 2	
3 0	
4 3	
1 2	
2 3	
3 1	

## Разбор задачи I. ICPC

В этой задаче необходимо научиться располагать ориентированный граф на круге так, чтобы все выходящие из вершины дуги следовали в непосредственно следующие за ней вершины, а все входящие в нее дуги следовали из вершин непосредственно предыдущих.

Будем говорить, что точка  $B$  на круге лежит между точками  $A$  и  $C$ , если при обходе круга по часовой стрелке, начиная от точки  $A$ , мы сначала встретим точку  $B$ , а затем точку  $C$ . Будем обозначать это при помощи традиционного знака  $<$ :  $A < B < C$ . Знак  $<$  предполагает, что все три точки  $A$ ,  $B$  и  $C$  различны, если же допустимо совпадение  $A = B$ ,  $A = C$  или даже  $A = B = C$ , будем обозначать это как  $A \leq B \leq C$ . Аналогичная запись допустима и для более чем 3-х точек, например,  $A < B < C < D$  означает, что при обходе круга от точки  $A$  по часовой стрелки будут встречены сначала точка  $B$ , затем  $C$  и затем  $D$ .

Для решения задачи понадобится несколько несложных наблюдений. Начнем с некоторой более простой (или удобной) формулировки того, что требуется сделать.

Лемма 1. Граф расположен на круге корректным образом.  $\Leftrightarrow$  Для каждой дуги графа  $A \rightarrow B$  и для любых вершин  $C$  и  $D$  таких, что

$A \leq C < D \leq B$ ,  $C \rightarrow D$  также является дугой графа.

Доказательство.

Утверждение достаточно очевидно, доказательство приводится только для полноты разбора.

$\Rightarrow$  Пусть график расположен корректным образом,  $A \rightarrow B$  — дуга графа и  $A \leq C < D \leq B$ ,  $C \rightarrow D$ . Так как  $D \leq B$ , то из корректности расположения и наличия дуги  $A \rightarrow B$  получаем наличие дуги  $A \rightarrow D$ . Так как  $A \leq C$ , то из корректности расположения и наличия дуги  $A \rightarrow D$  получаем наличие дуги  $C \rightarrow D$ .

$\Leftarrow$  Пусть для каждой дуги  $A \rightarrow B$  и для любых  $C$  и  $D$ ,  $A \leq C < D \leq B$ ,  $C \rightarrow D$  также является дугой. Полагая  $A = C$ , получаем, что дуги из  $A$  следуют в непосредственно следующие за  $A$  вершины по кругу. Полагая  $D = B$ , получаем, что дуги из  $D$  следуют в непосредственно предыдущие ей вершины по кругу. В силу произвольности  $A$  и  $D$  график расположен корректно.

Предположим, что график уже корректно расположен на круге. Для каждой вершины  $A$  рассмотрим непосредственно следующую за ней вершину  $next(A)$ . Возможны две ситуации:

1. Для некоторого  $A$ ,  $A \rightarrow next(A)$  не является дугой графа.
2. Для любого  $A$ ,  $A \rightarrow next(A)$  является дугой графа.

Рассмотрим решение для каждой из этих ситуаций по отдельности. Но прежде необходимо различить, какая из этих ситуаций имеет место для конкретного графа.

Теорема 1. Если график корректно расположен на круге, то в случае наличия хотя бы одного цикла в графике имеет место ситуация 2, а в случае отсутствия циклов — ситуация 1.

Доказательство.

Во второй ситуации график, очевидно, имеет цикл (более того, гамильтонов цикл). В самом деле, начиная с некоторой вершины  $A$  и последовательно переходя в  $next(A)$  по дуге, мы в итоге обойдем весь круг и вернемся в вершину  $A$ .

Справедливо и обратное, в случае наличия цикла имеет место вторая ситуация. Действительно, рассмотрим некоторый цикл. Начиная с произвольной вершины цикла, будемходить по кругу (по часовой стрелке) в следующую вершину цикла, пока не вернемся в исходную вершину. В результате этого процесса круг будет обойден целиком один или более раз. Отсюда следует, что для каждой вершины  $A$  существует дуга цикла  $X \rightarrow Y$  такая, что  $X \leq A < next(A) \leq Y$ . По лемме 1, отсюда  $A \rightarrow next(A)$  является дугой графа.

Таким образом, ситуация 1 может иметь место только в случае отсутствия циклов.

Проверка наличия цикла в графе может быть осуществлена за  $O(V+E)$ , в зависимости от ее результата мы можем переходить к рассмотрению 1-й или 2-й ситуации.

Начнем с 1-й ситуации, поскольку она несколько проще. С учетом отсутствия в графе некоторой дуги  $A \rightarrow \text{next}(A)$  любая дуга графа  $X \rightarrow Y$  такова, что  $\text{next}(A) \leq X < Y \leq A$ . Поэтому мы можем мысленно “разрезать” дугу  $A \rightarrow \text{next}(A)$  и получить, что граф расположен на отрезке (начинающимся с вершины  $\text{next}(A)$  и заканчивающимся вершиной  $A$ ; для определенности будем считать, что отрезок горизонтальный) таким образом, что все дуги из любой фиксированной вершины следуют в непосредственно примыкающие к ней справа вершины, а все дуги в любую фиксированную вершину следуют из непосредственно примыкающих к ней слева вершин. Далее до окончания рассмотрения 1-й ситуации будем считать, что граф расположен на отрезке.

В случае отрезка будем считать, что  $A < B$  обозначает, что вершина  $A$  находится левее вершины  $B$ . В случае допустимости их совпадения будем писать  $A \leq B$ . Нетрудно проверить, что для отрезка справедлив аналог леммы 1 (получаемый просто заменой в тексте леммы 1 слова “круг” на слово “отрезок”).

Рассмотрим соответствующий исходному графу неориентированный граф, получаемый заменой каждой дуги на неориентированное ребро. Назовем компонентами связности исходного графа компоненты связности его неориентированного варианта.

Теорема 2. Пусть граф корректно расположен на отрезке. Тогда каждая компонента связности расположена в подряд стоящих вершинах отрезка.

Доказательство.

Пусть некоторая компонента  $C_1$  расположена не в подряд стоящих вершинах отрезка. Пусть  $A$  — самая левая вершина компоненты  $C_1$ , а  $B$  — самая правая. Существует некоторая вершина  $X$ , принадлежащая некоторой другой компоненте  $C_2$  такая, что  $A < X < B$ .

Рассмотрим путь из вершины  $A$  в вершину  $B$  в неориентированном варианте графа. Хотя бы одно ребро  $C - D$  в этом пути таково, что  $C < X < D$ . Так как дуги в ориентированном варианте графа, относительно отрезка всегда идут слева направо, то в нем есть дуга  $C \rightarrow D$ . С учетом корректности расположения графа,  $C \rightarrow X$  также должна быть его дугой. Отсюда сразу следует, что  $X$  также принадлежит компоненте  $C_1$ . Полученное противоречие доказывает теорему.

Нетрудно видеть, что если граф корректно расположен на отрезке, то, с учетом того, что каждая компонента связности расположена в подряд

стоящих вершинах отрезка, порядок расположения компонент друг относительно друга может быть произвольным. Поэтому нам достаточно научиться располагать друг относительно друга вершины одной компоненты связности.

Так как каждая дуга  $A \rightarrow B$  графа относительно отрезка идет слева направо, то нас интересует топологическое упорядочивание вершин графа. В общем случае может существовать (экспоненциально) большое количество таких упорядочений, однако если компоненту связности можно корректно расположить на отрезке, то такое упорядочивание уникально.

Теорема 3. Если граф расположен на отрезке и неориентированный эквивалент графа связан, то в графе существует единственное топологическое упорядочивание.

Доказательство.

Нам достаточно доказать, что для любых вершин  $X$  и  $Y$  таких, что  $Y$  находится на отрезке непосредственно справа от  $X$ , в графе есть дуга  $X \rightarrow Y$ . Отсюда сразу следует, что единственное возможное топологическое упорядочивание — это тот порядок, в котором вершины сейчас находятся на отрезке.

Для доказательства, аналогично как в теореме 2, рассмотрим крайнюю слева вершину графа  $A$  и крайнюю справа вершину  $B$ . В (неориентированном) пути от  $A$  до  $B$  должно существовать ребро  $C - D$  такое, что  $C \leq X < Y \leq D$ . С учетом того, что  $C \rightarrow D$  является дугой графа, можно применить лемму 1 (для отрезка) и получить, что  $X \rightarrow Y$  также является дугой графа.

Таким образом, ситуация 1 рассмотрена полностью. Полный алгоритм для нее выглядит следующим образом:

- Выделить компоненты связности неориентированного варианта графа.
- В каждой компоненте связности построить топологическое упорядочивание.
- Расположить компоненты на отрезке в произвольном порядке, при этом внутри каждой компоненты располагая вершины согласно построенным топологическим упорядочиваниям.
- Проверить, является ли построенное расположение корректным. Если да, то решение найдено, иначе решения не существует.

Каждый из этих шагов можно выполнить за время  $O(V + E)$ , таким образом, полученный алгоритм линеен. Последний шаг несколько проще выполнить за время  $O(V + E \log E)$ , и это также допустимо.

Перейдем к рассмотрению 2-й ситуации. Ключом к успеху в 1-й ситуации было сведение задачи на круге к задаче на отрезке посредством “разрезания” круга по отсутствующей дуге. В этом случае все дуги на круге присутствуют, поэтому именно такой трюк повторить не получится. Поэтому поступим так: удалим одну вершину и некоторые дуги из графа так, чтобы в нем не стало циклов. Тогда для него будет иметь место 1-я ситуация, и мы можем найти нужный порядок следования вершин. Добавив удаленную вершину в нужное место на круге, мы получим решение для 2-й ситуации. Отметим, что описанный подход будет работать при выполнении двух условий:

- После устранения циклов решение для 1-й ситуации уникально. (В противном случае пришлось бы рассмотреть все решения, что ведет к увеличению сложности.)
- Мы можем определить, в какое место на круге следует вставить удаленную вершину. (В противном случае пришлось бы рассмотреть все  $O(V)$  вариантов вставки, что, опять же, ведет к увеличению сложности.)

К счастью, оба этих условия несложно выполнить, поэтому весь этот подход работает.

Рассмотрим детали. Выберем произвольную вершину  $A$  графа. Удалим из графа вершину  $A$  и все дуги  $X \rightarrow Y$  такие, что  $X \leq A \leq Y$ . Нетрудно видеть, что после такой операции в графе не останется циклов. В самом деле, можно рассмотреть произвольный цикл и начать обходить его с произвольной вершины вдоль круга, пока не вернемся в исходную вершину. Так как круг будет обойден целиком (1 или более раз), в некоторый момент мы будем находиться в вершине  $A$ . Для текущей дуги тогда будет выполнено  $X \leq A \leq Y$ .

Таким образом, такое преобразование вполне подходит нам, чтобы свести 2-ю ситуацию к 1-й. Однако непонятно, как его осуществить, так как проверка условия  $X \leq A \leq Y$  требует расположения графа на круге, которое мы еще только собираемся построить. Сначала заметим, что  $X \leq A \leq Y$  распадается на три условия:  $X = A$ ,  $Y = A$  или  $X < A < Y$ . Первые два тривиально проверяются, а ситуацию с 3-м условием разрешает следующая лемма.

Лемма 2. Пусть граф корректно расположен на круге. Тогда для любой дуги  $X \rightarrow Y$  условие  $X < A < Y$  выполнено тогда и только тогда, когда  $X \rightarrow A$  и  $A \rightarrow Y$  являются дугами графа.

Доказательство.

$\Rightarrow$  Пусть  $X < A < Y$  и  $X \rightarrow Y$  является дугой графа. Тогда в силу леммы 1 как  $X \rightarrow A$ , так и  $A \rightarrow Y$  являются дугами графа.

⇐ Пусть  $X \rightarrow Y$ ,  $X \rightarrow A$  и  $A \rightarrow Y$  являются дугами графа. Предположим, от противного,  $Y < A < X$ . Отсюда  $A < X < Y$ , значит, по лемме 1,  $A \rightarrow X$  также является дугой графа. Но по условию задачи граф не может содержать обе дуги  $X \rightarrow A$  и  $A \rightarrow X$ . Противоречие.

Таким образом, сведение 2-й ситуации к 1-й осуществлено. С учетом того, что  $A$  фиксировано, оно требует линейного времени  $O(V + E)$ .

Заметим, что в полученном в итоге графе  $V-1$  вершин и присутствуют  $V-2$  из исходных  $V$  дуг между соседними на круге вершинами. Отсюда получаем, что в неориентированном варианте графа есть только одна связная компонента, а значит, существует единственное решение — ее топологическое упорядочивание. Очевидно, что удаленную вершину  $A$  необходимо вставить сразу после самой последней вершины в этом топологическом упорядочении или непосредственно перед самой первой вершиной (с точки зрения цикличности круга в обоих случаях получаем один и тот же порядок).

Таким образом, обе ситуации рассмотрены. В итоге было получено линейное по времени и памяти решение задачи.

## Задача J. Numbers

Имя входного файла:	j.in
Имя выходного файла:	j.out
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Неотрицательное целое число называется К-значным, если его можно записать с помощью К штук цифр, а ( $K-1$ ) цифр для этого не достаточно. Например, 43 является 2-значным числом, 2010 — четырехзначное, а 0 и 5 — 1-значные числа.

Для данных А и В подсчитайте, сколько существует К-значных неотрицательных целых чисел в десятичной системе исчисления, где К не менее А и не более В.

### Ограничения

$$1 \leq A \leq B \leq 1000.$$

### Формат входного файла

Единственная строка содержит два целых числа А и В.

### Формат выходного файла

Выведите ответ задачи без ведущих нулей.

## Пример

<b>j.in</b>	<b>j.out</b>
1 1	10
3 5	99900

## Разбор задачи J. Numbers

Первым делом, посмотрим, сколько существует  $K$ -значных неотрицательных чисел для конкретного значения  $K$ . Для  $K = 1$  их, очевидно, 10 — все цифры от 0 до 9. Для  $K = 2$  нас интересуют все числа от 10 до 99 включительно, а всего их 90. Аналогично, для каждого значения  $K > 1$  нам интересны числа в интервале  $[10^{(K-1)}, 10^K - 1)$ , соответственно их количество  $10^K - 10^{(K-1)}$ .

На основе этого мы уже можем вычислить ответ, просуммировав количества  $K$ -значных чисел для каждого  $K$  от  $A$  до  $B$ , но возникает одна техническая сложность. Учитывая, что  $B$  ограничено сверху значением 1000, ответ может доходить до  $10^{1000}$ . Такое число без потери точности не может содержать ни один встроенный числовой тип данных. В общем случае, для хранения таких чисел используется концепция длинной арифметики. В нашем случае можно обойтись без неё.

Заметим, что если  $A > 1$ , то ответ является суммой разностей вида  $10^i - 10^{(i-1)} = 9 \cdot 10^{(i-1)}$ . Каждое из этих чисел соответствует написанию девятки в  $i$ -ом справа разряде десятичной записи ответа. Соответственно, ответ для конкретных  $A, B$  при  $A > 1$  — это  $B$ -значное число, у которого в старших разрядах, начиная с  $A$ -того справа, записаны 9-тки, а остальные цифры равны нулю. В случае же  $A = 1$  ответом являются все числа от 0 до  $10^B$ , их  $10^B$  штук, и поэтому ответ имеет вид 100...00, где количество нулей равно  $B$ . Зная вид ответа в обоих случаях, мы можем вывести его на экран без проведения каких-либо вычислений.

## Задача K. English

Имя входного файла:	<b>k.in</b>
Имя выходного файла:	<b>k.out</b>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

В английском языке 26 букв, из которых 5 (a, e, i, o, u) — гласные, 20 согласных, а одну считают полугласной, т.е. ее не считают ни гласной, ни согласной (буква ‘y’). Слово называется звучным, если в нём количество

гласных превосходит количество согласных. Если слово содержит некоторую букву более одного раза, то каждое её вхождение засчитывается — то есть, например, слово “alabama” звучное.

Для данного слова найдите, какое наименьшее количество букв в нём надо заменить, чтобы получить звучное слово.

## Ограничения

Длина данного слова не превосходит 1000 букв. Слово состоит из букв латинского алфавита в нижнем регистре.

## Формат входного файла

Единственная строка содержит данное слово.

## Формат выходного файла

Выведите минимальное количество букв, при замене которых данное слово становится звучным.

## Пример

<b>k.in</b>	<b>k.out</b>
dog	1
alabama	0
y	1

## Разбор задачи K. English

Подсчитаем количество гласных и согласных в данном слове. Обозначим их  $V$  (vowels) и  $C$  (consonants). До тех пор, пока  $V \leq C$ , нам надо менять какую-либо согласную букву на гласную, за счёт чего  $V$  увеличивается на 1, а  $C$  уменьшается. Количество таких замен можно найти и итерационно, и решив неравенство  $V + x > C - x$ , откуда получаем  $x > \frac{(C-V)}{2}$ , то есть ответом будет наименьшее неотрицательное число, превосходящее  $\frac{(C-V)}{2}$ . Единственный случай, когда этот подход не работает — это когда строка состоит только из букв ‘y’, то есть  $V = C = 0$ . Очевидно, тогда ответ равен 1.

## Задача L. Diamond

Имя входного файла:	l.in
Имя выходного файла:	l.out
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Вы — большой любитель ювелирных украшений и драгоценных камней. Недавно в ювелирном магазине напротив Вашего дома в продаже появился совершенно замечательный бриллиант. Цена у него, однако, тоже была совершенно замечательная — X долларов. С сожалением Вам пришлось констатировать, что для Вас это слишком дорого, как бы ни нравился сам камень.

Видимо, другие потенциальные покупатели пришли к тому же выводу, потому что через некоторое время магазин устроил рекламную акцию. “Получите 5%-ную скидку на бриллиант за каждую букву Вашего имени!” — гласил транспарант у входа в магазин. Формулировка, прямо скажем, не очень однозначная, поэтому Вы решили поговорить с продавцами в магазине. В результате разговора выяснилось:

- Если буква встречается в имени несколько раз, то она будет подсчитана только единожды. Т.е., например, в имени alexandra с точки зрения данной акции не 9, а 7 букв, т.к. буква a встречается трижды.
- 5%-ная скидка умножается на количество различных букв в имени. Т.е. в случае имени alexandra будет предоставлена 35%-ная скидка.
- Скидки более 100% не предоставляются, но ровно 100%-ая скидка может быть предоставлена (когда Вы задавали вопрос по этому поводу, продавцы смотрели на Вас как на полоумного).

Конечно, таким щедрым предложением грешно было не воспользоваться, но хотелось сэкономить еще. По счастливой случайности, недавно Вы познакомились с человеком, подделывающим паспорта. Услуги его оплачиваются следующим образом:

- Подделка паспорта стоит A долларов. В поддельном паспорте можно использовать Ваше настоящее имя без изменений, а можно изменить его посредством применения сколь угодно большого количества операций каждого из указанных дальше трех видов.
- Вставка одной буквы в любое место имени стоит B долларов.

- Удаление одной любой буквы из имени стоит С долларов. Ситуация, когда в результате удаления получается пустая строка, считается недопустимой.
- Изменение одной любой буквы из имени на любую другую букву английского алфавита стоит D долларов.

По заданным числам X, A, B, C, D, а также Вашему настоящему имени Name, определите минимальную денежную сумму, которую придется потратить на покупку бриллианта (считая как деньги, заплаченные собственно за бриллиант, так и деньги, потраченные на подделку паспорта). Подделывать паспорт не обязательно, Вы также можете просто предъявить Ваш настоящий паспорт с именем Name при покупке.

## Ограничения

Числа X, A, B, C и D — целые.

$1 \leq X, A, B, C, D \leq 1000000$ .

Name содержит от 1 до 15 символов.

Каждый символ в Name — буква английского алфавита в нижнем регистре.

## Формат входного файла

Первая строка входного файла содержит число X. Во второй строке записаны числа A, B, C и D, разделенные одиночными пробелами. В третьей строке записано Name.

## Формат выходного файла

Выведите минимальную денежную сумму, которую придется потратить на покупку бриллианта, измеренную в центах (в одном долларе 100 центов).

## Пример

1.in	1.out
100 1 2 3 4 eldar	3100
100 100 100 100 ivan	8000
9876 1 567 890 1 nebuchadnezzar	296780

## Разбор задачи L. Diamond

Посчитаем количество различных букв и количество повторяющихся букв в данном имени. Вычислим, в какую сумму бриллиант обойдётся без подделки паспорта и перейдём к рассмотрению махинаций.

Во-первых, сама подделка паспорта уже обходится в  $A$  долларов, что мы добавим к первоначальной стоимости. Дальше нам надо решить, какие операции произвести. Сразу отметим, что удаление буквы не может увеличить количество различных букв, поэтому эта операция бесполезна. Также отметим, что увеличивать количество различных букв в имени после 20 уже не имеет смысла: скидок больше 100% не предоставляется.

Теперь сравним цены за добавление и за изменение буквы. Если первая операция обходится дешевле, то вторая нам не потребуется вообще. Тогда надо определить, выгодно ли добавление буквы в имя (то есть меньше ли эти затраты выигрыша в скидке), и в таком случае добавить столько букв, чтобы количество различных стало 20. Если же замена буквы обходится дешевле добавления, то в первую очередь надо использовать замены повторяющихся букв на новые, а потом уже перейти к добавлениям. Если получившаяся в итоге стоимость меньше изначальной, то это ответ, в противном случае ответ — цена бриллианта, если покупать его с подлинным паспортом.

## Задача M. Puzzle

Имя входного файла:	<code>m.in</code>
Имя выходного файла:	<code>m.out</code>
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Задано  $N^2$  десятичных цифр от 1 до 9.

Рассмотрим размещение этих цифр в клетках квадратной таблицы  $N \times N$ , по одной цифре в клетке. В каждой из строк таблицы, читая слева направо, получим десятичную запись некоторого  $N$ -значного числа. В каждом из столбцов таблицы, читая сверху вниз, также получим десятичную запись некоторого  $N$ -значного числа. Пусть  $S$  — это сумма всех  $N$  чисел по строкам и всех  $N$  чисел по столбцам.

Разместите числа внутри таблицы таким образом, чтобы значение  $S$  было максимальным.

## Ограничения

Число  $N$  целое.

$1 \leq N \leq 8$ .

### **Формат входного файла**

Первая строка входного файла содержит целое число  $N$ . Во второй строке записаны без разделителей  $N^2$  десятичных цифр от 1 до 9.

### **Формат выходного файла**

Выведите максимально возможное значение суммы  $S$ .

### **Пример**

<b>m.in</b>	<b>m.out</b>
2	303
9174	

### **Разбор задачи M. Puzzle**

Рассмотрим, какой вклад в итоговую сумму вносит цифра в клетке  $(i, j)$ . В записанном по вертикали числе эта цифра будет слева  $i$ -ой, соответственно, она умножится на коэффициент  $10^{N-i}$ . В записанном по горизонтали числе она, по аналогичной логике, помножиться на коэффициент  $10^{N-j}$ . Таким образом, мы можем для каждой клетки таблицы определить коэффициент, с которым входит записанная в ней цифра в итоговую сумму.

Очевидно, в клетку, соответствующую максимальному коэффициенту, нужно записать максимальное число. В клетку со следующим максимальным коэффициентом надо записать следующее максимальное число, и так далее. Поэтому, чтобы найти максимальную возможную сумму, достаточно отсортировать коэффициенты и цифры в порядке возрастания и посчитать сумму произведений элементов, оказавшихся на одинаковых позициях.

## День восьмой (19.02.2011 г.) Конкурс Митричева Петра Игоревича

### Об авторе...

**Петр Митричев**, родился в 1985 году в Москве. Окончил механико-математический факультет Московского Государственного Университета. Работает инженером в компании “Google”.

Основные достижения:

- трехкратный медалист международных олимпиад школьников по информатике, победитель Всероссийской олимпиады школьников по информатике 2000, 2001 и 2002 гг;
- победитель в составе команды МГУ NEERC-2002 и серебряный призер NEERC-2004;
- участник команд МГУ, ставших вице-чемпионами на студенческих командных чемпионатах мира ACM ICPC 2003 и 2005 гг;
- в 2006-м выиграл открытые командные соревнования на кубок СБООС – МГУ, причем его команду представлял только один участник – он сам;
- установил уникальное достижение: он выиграл и TopCoder Open-2006, и Google Code Jam-2006, и TopCoder Collegiate Challenge-2006, таким образом, став первым в истории участником, которому удалось в один год одержать победу во всех трёх крупнейших мировых индивидуальных турнирах по спортивному программированию;
- два года подряд становился победителем финала TopCoder Collegiate Challenge Algorithm (2006 и 2007);
- занял второе место на TopCoder Open 2008;
- был одним из тренеров команды МГУ, занявшей второе место на студенческом командном чемпионате мира по программированию 2010 года и завоевавшей звание чемпионов Европы.



## Теоретический материал. Задачи “от $L$ до $R$ ”

Вот простая задача такого типа:

### Задача

Сколько существует целых чисел  $X$  между  $L$  и  $R$  включительно таких, что их сумма цифр не превышает  $A$ ?  $1 \leq L \leq R \leq 10^{18}$ .

### Решение

Для решения этой задачи выполним несколько стандартных шагов. Первый шаг: вместо того, чтобы решать задачу для отрезка  $[L, R]$ , решим ее для отрезков  $[0, R]$  и  $[0, L - 1]$ , а ответ построим вычитанием.

Второй шаг: решаем задачу для отрезка  $[0, R]$  — пусть функция, находящая решение называется  $solve(R, A)$ . Пусть в числе  $R k$  цифр:  $10^{k-1} \leq R < 10^k$ . Разобьем отрезок  $[0, R]$  на несколько отрезков в зависимости от  $k$ -й с конца цифры:  $[0, 10^{k-1} - 1]$ ,  $[10^{k-1}, 2 \cdot 10^{k-1} - 1]$ ,  $\dots$ ,  $[d \cdot 10^{k-1}, R]$ .

Легко видеть, что наша задача разбилась на аналогичные подзадачи — зачеркнув зафиксированную  $k$ -ю с конца цифру, мы получили новый отрезок вида  $[0, R']$  и новую максимально возможную сумму цифр. Иными словами,  $solve(R, A) = solve(10^{k-1} - 1, A) + solve(10^{k-1} - 1, A - 1) + \dots + solve(R - d \cdot 10^{k-1}, A - d)$ .

Задача, которая сводится к аналогичным, но меньшим подзадачам — это основа для применения динамического программирования. В данном случае параметрами являются два числа  $R$  и  $A$ . Но  $R$  принимает значения аж до  $10^{18}$ , так что завести массив такого размера не получится. Но это и не нужно: легко видеть, что достижимых значений  $R$  очень мало. Это числа, состоящие из одних девяток, и числа, являющиеся суффиксами исходного числа  $R$ .

Поэтому мы либо должны пронумеровать эти достижимые числа, чтобы запоминать вычисленные ответы для подзадач по их индексу, либо просто использовать структуру данных, которая хранит вычисленные ответы только для нужных пар — хеш-таблицу или сбалансированное дерево.

### Более сложная задача

Сколько существует целых чисел  $X$  между  $L$  и  $R$  включительно таких, что если их цифры записать в обратном порядке, полученное число также лежит между  $L$  и  $R$ ?  $1 \leq L \leq R \leq 10^{18}$ .

### Решение

Здесь первый шаг изложенного выше решения не проходит, поэтому будем в дальнейшем решении использовать и  $L$ , и  $R$  (упражнение для читателя: а может быть, все-таки проходит?).

Опять обозначим через  $k$  количество цифр в  $R$  и посмотрим на  $k$ -ю с конца цифру нашего числа. В развернутом числе это будет последняя цифра. Запишем наше число в виде  $d \cdot 10^{k-1} + A$ , и обозначим за  $\text{rev}(x, k)$  функцию разворота цифр в числе  $x$  длины  $k$  (возможно, дополненном ведущими нулями).

Тогда  $\text{rev}(X) = \text{rev}(d \cdot 10^{k-1} + A) = \text{rev}(A) \cdot 10 + d$ . Условие, которое нам необходимо проверить, записывается в виде  $L \leq \text{rev}(A) \cdot 10 + d \leq R$ , что преобразуется к  $\frac{L-d}{10} \leq \text{rev}(A) \leq \frac{R-d}{10}$ .

С первого взгляда кажется, что мы зашли в тупик: полученная подзадача не эквивалентна исходной. Однако, мы можем использовать стандартный трюк динамического программирования: добавим параметров к задаче! Будем искать количество чисел на отрезке  $[L, R]$  таких, что если их цифры записать в обратном порядке, полученное число лежит на другом отрезке  $[P, Q]$ .

Легко видеть, что для такой задачи описанная выше операция “разбития” отрезка  $[L, R]$  по первой цифре сведет ее к нескольким меньшим подзадачам такого же типа, а значит, можно применить динамическое программирование!

Правда, как и в первой рассмотренной задаче, количество состояний поначалу потрясает – 4 числа, каждое до  $10^{18}$ . Но внимательное изучение формул, написанных выше, помогает понять, что границы первого отрезка всегда будут либо от 0 до  $10^k - 1$ , либо от суффикса одного из чисел  $L$  или  $R$  до 0 или  $10^k - 1$ , а границами второго отрезка всегда будут префиксы чисел  $L$  и  $R$ , возможно, уменьшенные на 1. Поэтому всего достижимых состояний  $\text{poly}(18)$ . Вот ещё одна задача, решаемая аналогичной техникой:

### **Задача для самостоятельного решения**

Сколько есть чисел между  $L$  и  $R$ , являющихся палиндромами и делящихся на 7?

А вот задача, для решения которой применяется другая техника:

### **Задача**

Сколько есть чисел между  $L$  и  $R$ , не делящихся ни на одно из заданных простых чисел  $p_1, p_2, \dots, p_k$ ?  $1 \leq L \leq R \leq 10^{12}$ ,  $1 \leq k \leq 30$ .

### **Решение**

Базовой идеей для решения этой задачи может служить такая: всего между  $L$  и  $R$   $R - L + 1$  число. Из них  $\lfloor \frac{R}{p_1} \rfloor - \lfloor \frac{L-1}{p_1} \rfloor$  делятся на  $p_1$ , значит их надо вычесть. Аналогично, вычитаем числа делящиеся на  $p_2$  и так далее — и получаем неверный ответ! Потому что числа, делящиеся и на  $p_1$ , и на  $p_2$ , мы вычли два раза — значит, теперь их нужно один раз прибавить. И так далее.

Общая формулировка такого процесса называется *формулой включений-исключений*: Пусть есть некоторое множество  $X$  и набор его подмножеств  $A_1, A_2, \dots$ . Тогда количество элементов в объединении  $A_i$  есть сумма количеств элементов в  $A_i$ , минус сумма количеств элементов в попарных пересечениях  $A_i$ , плюс сумма количеств элементов в тройных пересечениях  $A_i$ , и так далее.

В конкретном случае делимости, эту теорему можно переформулировать, используя *функцию Мёбиуса*:  $\mu(n)$  равно 0, если  $n$  делится на квадрат какого-либо простого числа, 1, если все простые числа входят в разложение  $n$  в первой степени и их количество чётно, и -1, если нечётно.

Формула включений-исключений в этом случае становится *формулой обращения Мёбиуса*:  $g(n) = \sum_{d|n} f(d)$  тогда и только тогда, когда

$$f(n) = \sum_{d|n} \mu(d) \cdot g(n/d).$$

Как эта формула применима в нашей задаче? Обозначим за  $N$  произведение всех простых чисел  $p_i$ . Обозначим за  $g(n)$  количество чисел от  $L$  до  $R$ , делящихся на  $N/n$ , а за  $f(n)$  количество чисел от  $L$  до  $R$ , НОД которых с  $N$  равен  $N/n$ . Тогда нетрудно видеть, что  $g(n)$  можно разбить на сумму  $f(n')$ , где  $n'$  делит  $n$ . Тем самым  $f(n)$  выражается через  $g(n)$  через формулу Мёбиуса, а ведь  $f(N)$  — ответ в нашей задаче.

Возвращаясь к самой задаче, нужно ещё отметить, что на первый взгляд кажется, что решения, приведенные выше, работают за  $O(2^k)$ , так как им нужно рассматривать всевозможные произведения  $p_i$ . Однако нетрудно видеть, что нас интересуют лишь произведения, не превосходящие  $R$ , а таких мало.

Из приведенного выше решения видно, что на функцию Мёбиуса можно добавлять дополнительные навороты:

### Задача

Сколько есть чисел между  $L$  и  $R$ , свободных от квадратов (не делящихся на квадрат никакого простого числа)?  $1 \leq L \leq R \leq 10^{12}$ .

### Решение

Так как максимальный квадрат, на который может делиться такое число, это квадрат миллиона, то возникает желание просто перебрать их все, но что делать потом? Тут нам на выручку опять приходит функция Мёбиуса!

Пусть  $g(n)$  — это количество чисел между  $L$  и  $R$ , делящихся на  $n^2$ , а  $f(n)$  — это количество чисел между  $L$  и  $R$  таких, что *максимальный квадрат*, на который они делятся — это  $n^2$ .

Тогда нетрудно видеть, что  $g(n)$  — это сумма  $f(n')$  для всех  $n'$  делящихся на  $n$ . Для формулы Мёбиуса нужно, чтобы, наоборот,  $n'$  делило

*n*. Как и в предыдущей задаче, это достигается заменой *n* на  $N/n$  (где *N*, скажем, произведение всех чисел от 1 до миллиона), и мы опять можем применить формулу Мёбиуса: ответом будет сумма по всем *n* от 1 до миллиона  $\mu(n) \cdot (\lfloor \frac{R}{n^2} \rfloor - \lfloor \frac{L-1}{n^2} \rfloor)$ .

## Задачи и разборы

### Задача А. Automatic Input Verifier

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

Tons and tons of programming contest problems contain integers separated with spaces and newlines in the input. In many cases, the amount of numbers found inside the file and in each particular line is determined by the numbers themselves.

More formally, we define an *input description* as a sequence of statements of the following types:

1. “The first/next line contains *number* non-negative integer(s), *variable*<sub>1</sub>, *variable*<sub>2</sub>, ..., *variable*<sub>*number*</sub>.”. Here, *number* is a positive integer between 1 and 3, and *variables* are arbitrary distinct variable names that have never appeared in the input description before this statement. When *number* is equal to 1, “integer(s)” should be replaced by “integer”, otherwise it should be replaced by “integers”.
2. “The first/next line contains *variable* integers.”. Here, *variable* is a variable name that has appeared in this description before this statement.
3. “The first/next line contains a non-negative integer *variable*, followed by *variable* integers.”. Here, *variable* is a variable name that has never appeared in the input description before this statement, and will never appear in the input description after this statement. Please note that the same variable name is repeated twice in this statement.
4. “The first/next line contains *number* integer(s).”. Here, *number* is a positive integer. When *number* is equal to 1,

“`integer(s)`” should be replaced by “`integer`”, otherwise it should be replaced by “`integers`”.

5. “The first/next *variable* lines contain *number integer(s) each*.”. Here, *variable* is a variable name that has appeared in this description before this statement, and *number* is a positive integer. When *number* is equal to 1, “`integer(s)`” should be replaced by “`integer`”, otherwise it should be replaced by “`integers`”.
6. “The first/next *variable<sub>1</sub>* lines contain *variable<sub>2</sub> integers each*.”. Here, *variable<sub>1</sub>* and *variable<sub>2</sub>* are (possibly the same) variable names that have appeared in this description before this statement.
7. “The first/next *variable<sub>1</sub>* lines contain a non-negative integer *variable<sub>2</sub>*, followed by *variable<sub>2</sub> integers each*.”. Here, *variable<sub>1</sub>* is a variable name that has appeared in this description before this statement. *variable<sub>2</sub>* is a variable name that has never appeared in this description before this statement, and will never appear in this description after this statement, and is different from *variable<sub>1</sub>*. Please note that, unlike all other variables, *variable<sub>2</sub>* may have different values for each line described by this statement. This is useful, for example, for listing several heaps with varied amount of items in each heap.

Each occurrence of “`first/next`” in the above statements should be substituted by “`first`” in the first statement of the input description, and by “`next`” in all other statements.

Given several input files, you need to verify whether there exists an input description containing not more than 4 statements that fits most of the given input files, and highlight the input files that don’t fit this description.

## Формат входного файла

The first line of the input file contains an integer  $n$ ,  $1 \leq n \leq 10$ , the number of input files given. Please note, that the meaning of words “input file” below corresponds to the second meaning of that term in this paragraph, not the first one.

The description of each given input file starts with a line containing a dash (“-”). The actual given input file follows, with each line appended a exclamation mark (“!”) to avoid difficulties in processing empty lines.

After excluding the exclamation mark at the end, each line of each given input file will contain between 0 and 1000 integers, each between -1000 and

1000, separated with single spaces. Each given input file will contain at most 10000 lines and at most 100000 integers.

## **Формат выходного файла**

In case there exists an input description with not more than 4 statements describing all given input files, output “**All good!**” (without quotes) to the first line of the output file, followed by the input description itself, one statement per line. Use non-empty strings of at most 10 lowercase English letters to denote variables. In case there are several possible descriptions, output any.

In case there exists an input description with not more than 4 statements describing at least one given input file, find any such description that describes the most given input files. Output “**Bad format: numbers.**” (without quotes) to the first line of the output file, where *numbers* is a single-space separated list of input file numbers that don’t fit the given input description. The input files are numbered from 1 to *n* in the order they are given. Then output the input description itself.

In case no input description with not more than 4 statements fits any given input file, output “**FAIL.**” to the only line of the output file.

## Примеры

a.in	a.out
3 - 2 3! 1 2! 3 5! -100 24! - 5 0! - 2 2! 2 2! 2 2!	All good! The first line contains 2 non-negative integers, v, e. The next e lines contain 2 integers each.
4 - 0! 1 0! ! - 2 1 2! 2 0! ! ! - 4 -100 -100 0 100! 3 0! ! ! ! ! - 3 3 3 3! 3 1! ! ! !	Bad format: 4 3. The first line contains a non-negative integer c, followed by c integers. The next line contains 2 non-negative integers, number, line. The next number lines contain line integers each.
1 - -1000! -1000 -1000! -1000 -1000 -1000! -1000 -1000 -1000 -1000! -1000 -1000 -1000 -1000 -1000!	FAIL.

## Разбор задачи A. Automatic Input Verifier

В задаче просто нужно аккуратно реализовать то, что написано в условии.

## Задача B. Friendly Points

Имя входного файла:	b.in
Имя выходного файла:	b.out
Ограничение по времени:	12 с
Ограничение по памяти:	256 Мб

Consider  $n$  distinct points on a plane.

Two points from that set are said to be *friends*, when there exists a rectangle with sides parallel to coordinate axes that contains those two points and doesn't contain any other point from the given set. A rectangle is said to *contain* a point if the point lies within the rectangle or on its border.

How many pairs of friends are there among the given points?

### Формат входного файла

The first line of the input file contains an integer  $n$ ,  $1 \leq n \leq 100\,000$ .

The next  $n$  lines contain two integers each, the coordinates of the given points. The coordinates don't exceed  $10^9$  by absolute value.

### Формат выходного файла

Output one integer number — the sought number of pairs of friends.

### Примеры

b.in	b.out
5	
0 0	
0 2	
2 0	
2 2	
1 1	

## Разбор задачи B. Friendly Points

Сначала разобьем ответ на четыре класса: когда вектор от одной точки до второй идет вправо-вверх, вправо-вниз, строго вправо, строго вверх.

Последние два случая очень просты, а первые два аналогичны, поэтому будем решать только первый.

Будем решать задачу так: будем идти по точкам сверху вниз, тогда для каждой очередной точки нужно прибавить к ответу следующую величину: количество “инкрементальных минимумов” справа от нее. Более формально, если мы будем идти слева направо, то нужно посчитать те точки, которые оказываются ниже всех предыдущих рассмотренных.

Это можно сделать с помощью специального дерева интервалов, в котором каждая вершина содержит количество “инкрементальных минимумов” на отрезке, ей соответствующем, и значение последнего из них (то есть просто минимум на отрезке).

### Задача C. Expanding Lake

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

You've been walking around your garden when rain started. At first, you were feeling relieved because it was the first rain in two months. But suddenly you've realized that this is not just a normal rain, but the all-drowning rain the news were talking about. So now you need to get home *fast*.

Imagine your garden as an infinite plane. You are at point  $(x_0, y_0)$  on this plane, while your home is at point  $(0, 0)$ . You can move with speed  $v_0$ . Easy:

$$\frac{\sqrt{x_0^2 + y_0^2}}{v_0},$$

you might think.

But there's also a dreadful lake in your garden. It is a circle centered at  $(x_1, y_1)$  and radius  $r_1$ . Moreover, because of the rain, it is expanding with speed  $v_1$ , so its radius after time  $t$  is equal to  $r_1 + v_1 t$  (its center doesn't move).

You can't go into the lake as you're suddenly very afraid of water. It's OK for you to stand on the border of the lake.

Is it possible for you to get home? If yes, what is the minimum time required to do that?

#### Формат входного файла

The first line of the input file contains three integers  $x_0$ ,  $y_0$  and  $v_0$ ,  $x_0^2 + y_0^2 > 0$ ,  $-1000 \leq x_0, y_0 \leq 1000$ ,  $1 \leq v_0 \leq 1000$ .

The second line of the input file contains four integers  $x_1$ ,  $y_1$ ,  $r_1$  and  $v_1$ ,  
 $x_1^2 + y_1^2 > r_1^2$ ,  $-1000 \leq x_1, y_1 \leq 1000$ ,  $1 \leq r_1, v_1 \leq 1000$ ,  
 $(x_0 - x_1)^2 + (y_0 - y_1)^2 > r_1^2$ .

It is guaranteed that in case it's possible to get home, it would still be possible when the initial radius of the lake is  $r_1 + 10^{-3}$ ; in case it's impossible to get home, it would still be impossible when the initial radius of the lake is  $r_1 - 10^{-3}$ .

### Формат выходного файла

Output one floating-point number denoting the minimum time required to get home. Your answer will be considered correct, if it's within  $10^{-9}$  relative or absolute error of the correct one.

In case you can't get home, output -1.

### Примеры

c.in	c.out
10 0 3 7 0 1 1	3.753280475952816
10 0 2 7 0 1 1	-1

### Разбор задачи C. Expanding Lake

Задача, в основном, по математике. Как и в других аналогичных задачах, мы либо идем по прямой, либо идем по “касательной” до озера, потом вдоль озера, потом по “касательной” от озера. Единственная разница у нас в определении “касательной” — для этого надо записать дифференциальное уравнение нашего движения вдоль озера, найти формулу для нашей траектории и получить направление касательной в каждой точке. Важно понять, что это не касательные к самому озеру в данный момент времени, а к “спирали”, по которой мы движемся вдоль озера.

### Задача D. Octahedron And Dominoes

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

Andrew has just made a breakthrough in puzzle making: he's invented a new puzzle that is both simple to understand and unlike everything, that has been invented before.

His new puzzle looks like an octahedron with each of its (triangular) faces split into  $n^2$  smaller triangles by  $3(n - 1)$  lines,  $n - 1$  parallel to each side of the face.

Some of smaller triangles are black and some are white. The objective is to cover all white triangles with triangular dominoes without overlapping, leaving all black triangles uncovered. A triangular domino is two small triangles with one common side. Please note that a domino can cover one small triangle on a side of the octahedron and another on the adjacent side of the octahedron (in other words, you can bend your domino in the middle). You must place each domino so that it covers exactly two white small triangles (that means you can't cover some part of a triangle with one domino and the rest with another domino, for example).

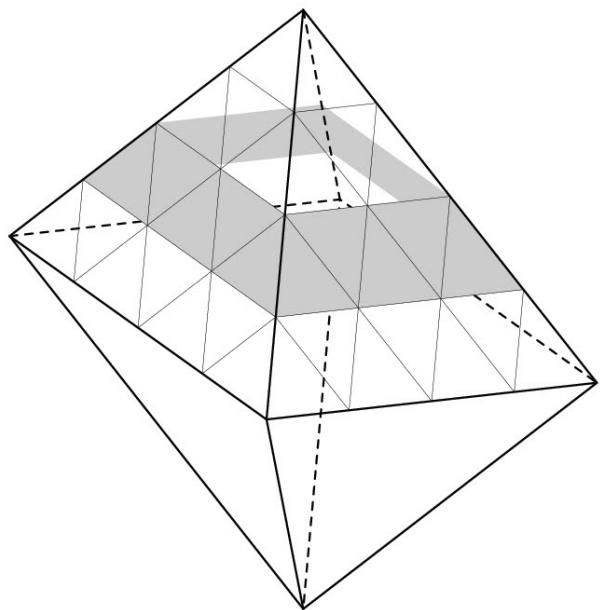
How many solutions does the given puzzle have?

### Формат входного файла

The first line of the input file contains an integer  $n$  ( $1 \leq n \leq 4$ ).

The next  $2n$  lines describe, which small triangles are black and which are white. Imagine the octahedron standing on its vertex, with one of its other vertices pointing on us. Then the small triangles can be split into horizontal layers. The topmost layer contains 4 triangles, one per each side that ends in the top vertex. The next layer contains 12 triangles, 3 per each side that ends in the top vertex, and so on. The size of each layer increases by 8 until it reaches  $8n - 4$  in the middle, then the next layer is again of  $8n - 4$  small triangles (here we switched from the top 4 faces to the bottom 4 faces), and then the size of each next layer is less by 8 until we reach the bottommost layer that has just 4 triangles.

The octrahedron is given layer-by-layer, and each layer is started with the leftmost small triangle on the visible part of the octahedron (the four “front” faces), goes from left to right along the visible part, and then switches to the



invisible part and goes back from right to left. One layer is highlighted on the picture above.

Each small triangle is specified either by ‘.’, denoting a white triangle, or by ‘\*’, denoting a black triangle.

### **Формат выходного файла**

Output one integer — the number of possible coverings of white triangles of the given octahedron with triangular dominoes.

### **Пример**

<b>d.in</b>	<b>d.out</b>
2 . * ** ..... * ... ***** * * * * * * ....	2
3 .... ***** * * * * * * ..... . . . . . . . . ***** * * * * * * * * * * * * .... ****	8

### **Разбор задачи D. Octahedron And Dominoes**

Сначала построим граф белых треугольников (соединены соседние). Затем динамическое программирование по профилю: рассматриваем вершины по одной и для всех рассмотренных вершин, для которых есть еще хотя бы одна соседняя нерассмотренная, помним, покрыты они или нет. Нужно рассматривать вершины в таком порядке, чтобы количество таких “промежуточных” вершин в каждый момент времени было невелико. Например, в том порядке, в котором они заданы во входном файле.

Решение за  $O(\text{числотреугольников} \cdot 2^{\text{число “промежуточных” треугольников}})$ . Максимальное число “промежуточных” треугольников —  $4n+1$  или около того.

## Задача E. Tiny Puzzle

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

You've decided to make a puzzle as a part of the "re-branding" campaign run by your company. In the box this puzzle will look as the new logo of your company, that is, it will be a set of 9 (nine) connected unit squares on a rectangular grid. However, these 9 squares will not necessarily all be glued together, i.e., they may form several connected parts (however, you're not allowed to split squares in the middle). The objective of the puzzle will be to rearrange those parts so that they form a  $3 \times 3$  square. When rearranging, one is only allowed to shift parts, and NOT to rotate or to flip them.

To make the puzzle more challenging, you need to have as few parts as possible. Given the logo, compute how many parts do you need and how to rearrange them into a  $3 \times 3$  square.

Every word "connected" in the problem statement means connectedness in the side-by-side sense.

### Формат входного файла

The first line of the input file contains two integers  $H$  and  $W$ . The next  $H$  lines will contain  $W$  characters each, describing your logo. Each of those characters will be either '.' (dot), denoting an empty square, or 'X' (capital English letter X), denoting a square occupied by the logo. All the 'X's will form a connected figure, there will be exactly 9 'X's in the input. The first line, the last line, the first column and the last column will contain at least one 'X', in other words, there will not be unnecessary empty rows at any side of the grid.

### Формат выходного файла

On the first line, output the required number of parts  $K$ . Afterwards, output one of the possible rearrangements, more precisely: on the next  $H$  lines output the logo with all the 'X's replaced by the first  $K$  capital letters of English alphabet, each letter denoting one part. Then output an empty line, and then 3 lines containing 3 characters each, containing the same  $K$  parts but shifted around so that they form a  $3 \times 3$  square.

All the parts must be connected, and  $K$  must be as little as possible. If there're several possible solutions, output any.

## Пример

<b>e.in</b>	<b>e.out</b>
4 5	3
....X	....A
....X	....C
..XXX	..CCC
XXXX.	BBCC.
	BBC
	CCC
	CCA

## Разбор задачи E. Tiny Puzzle

Задача решается любым разумным перебором всех вариантов.

## Задача F. Circular Island

Имя входного файла:	<b>f.in</b>
Имя выходного файла:	<b>f.out</b>
Ограничение по времени:	6 с
Ограничение по памяти:	256 Мб

Andrew has just made a breakthrough in geography: he has found an island that was previously unknown to the civilized world. This island has a shape of perfect circle and it is inhabited by two tribes, *Java* and *Seepplusplus*.

After a brief contact with the aborigines, Andrew found out that the boundary between the lands of the tribes is a straight line. Moreover, he knows the locations of several Java villages and of several Seepplusplus villages (each is of course located within or on the boundary of the corresponding tribe's land).

Now he needs to find out what is the minimal and maximal possible area of Java's land. Help him!

### Формат входного файла

The first line of the input file contains one integer  $r$  — the radius of the island ( $1 \leq r \leq 10^9$ ).

The next line contains one integer  $n$  ( $1 \leq n \leq 50\,000$ ) — the number of Java villages. Each of the next  $n$  lines contains two integers  $x$  and  $y$  — the coordinates of Java villages.

The next line contains one integer  $m$  ( $1 \leq m \leq 50\,000$ ) — the number of Seeplusplus villages. Each of the next  $m$  lines contains two integers  $x$  and  $y$  — the coordinates of Seeplusplus villages.

The center of the island has coordinates  $(0, 0)$ , each village is within the island and at least  $r/10$  away from the island boundary. No two villages coincide. The input is guaranteed to be valid — there will always be at least one straight line separating the Java villages from the Seeplusplus villages.

### Формат выходного файла

Output two floating-point numbers, separated with a space — the minimal and maximal possible area of the Java land. An area will be considered correct if it is within  $10^{-6}$  relative error of the right answer.

### Пример

<b>f.in</b>	<b>f.out</b>
6	12.389928320447176
2	56.548667764616276
3 4	
-3 4	
1	
0 0	

### Разбор задачи F. Circular Island

Отбросим точки каждого типа, не являющиеся вершинами выпуклой оболочки точек этого типа. Далее, поворачиваем разделяющую прямую и помним, какие точки являются крайними в каждой из двух выпуклых оболочек. Минимум и максимум площади достигаются в одном из трех случаев:

- 1) когда меняется одна из крайних точек;
- 2) когда прямая, соединяющая какую-то крайнюю точку первого типа и какую-то крайнюю точку второго типа, параллельна текущей разделяющей прямой;
- 3) когда прямая, проходящая через крайнюю точку и центр круга, перпендикулярна или параллельна текущей разделяющей прямой.

Решение за  $O(n \log n)$ .

## Задача G. Traffic Jam

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

The most annoying thing in Moscow traffic jams is that drivers constantly try to suddenly change lanes in order to move faster. In this problem you'll have to find out whether this is a reasonable strategy or not.

We'll study a relatively simple mathematical model of a traffic jam. Assume that there's an  $N$ -lane road, lanes numbered from 1 to  $N$ , and  $i^{th}$  lane is moving with speed  $b_i + a_i \cdot \sin(t + \delta_i)$  at the moment  $t$ . It is always true that  $b_i > a_i$ , i.e., the speed of movement is always positive. You can change the lane you're in at any time, it takes  $c \cdot |x - y|$  to change from  $x^{th}$  lane to  $y^{th}$ . We'll assume that you're not moving forward during that period.

Determine the time you need to travel the distance of  $d$ , and the method of achieving that time. You're starting at the moment 0 at lane 1, you may finish at any lane.

### Формат входного файла

The first line of the input file contains two integers  $N$  and  $d$  and a floating-point number  $c$ ,  $1 \leq N \leq 5$ ,  $1 \leq d \leq 1000$ ,  $0.001 \leq c \leq 1000$ . The next  $N$  lines describe lanes, each containing two integers  $a_i$  and  $b_i$  and a floating-point number  $\delta_i$ ,  $0 \leq a_i < b_i \leq 100$ ,  $0 \leq \delta_i < 2\pi$ .

### Формат выходного файла

On the first line of output print the minimal time required to travel the distance of  $d$ . On the second line of output print the number  $K$  of lane changes required to do that.  $K$  should not be more than  $10^6$ , it is guaranteed that there always exists an optimal strategy requiring not more than  $10^6$  lane changes. On the next  $K$  lines print the changes themselves, each line should contain the new lane number and the time when the change is started. The changes should be printed in chronological order. If there're many possible schedules, output any.

Output all the floating-point numbers with maximal precision possible. Your solution will be considered correct if verifying your schedule doesn't lead to discrepancies of more than  $10^{-6}$  (in checking the total distance traveled, in checking that lane changes are non-overlapping, etc), so it's better for you to output at least 10-12 digits after the decimal point in each floating-point number.

## Пример

<b>g.in</b>	<b>g.out</b>
1 100 0.5	19.71726232777025
4 5 0	0
3 100 0.5	19.052103083697858
4 5 0	4
2 5 0.5	2 3.6645304897691258
0 5 0	1 5.783185307179586 2 9.947715796948712 3 15.207963267948966

## Разбор задачи G. Traffic Jam

Если бы менять полосы можно было только в целые моменты времени, задача бы решалась стандартным динамическим программированием — какое максимальное расстояние можно проехать, оказавшись в момент времени  $t$  на полосе  $i$ .

Однако, непрерывность возможных моментов смены полосы можно побороть: нетрудно видеть, что перестраиваться с полосы  $i$  на полосу  $j$  имеет смысл либо в момент времени 0, либо в такой момент времени, что скорость полосы  $i$  в момент начала перестройки равна скорости полосы  $j$  в момент конца перестройки — а все такие моменты находятся путем решения простого тригонометрического уравнения.

## Задача H. Number Permutations

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

Two distinct positive integer numbers with decimal notations of the same length are called *similar*, if their decimal notations can be obtained from each other by permuting the digits.

How many numbers from the segment  $[l, r]$  have exactly one similar number in that segment?

## Формат входного файла

The input file contains two integer numbers  $l$  and  $r$  ( $1 \leq l \leq r \leq 10^{15}$ ).

## Формат выходного файла

Output one integer number — the sought amount.

## Пример

<b>h.in</b>	<b>h.out</b>
10 99	72

## Разбор задачи H. Number Permutations

Основное соображение, необходимое для решения этой задачи: если наше число находится достаточно далеко от границ, и у него не все три последние цифры одинаковые, то найдется хотя бы три перестановки последних цифр, дающие число внутри границ. Поэтому наши числа нужно искать среди тех, которые получаются дописыванием к префиксу левой или правой границы, измененному не более чем на 1000, нескольких одинаковых цифр.

## Задача I. Perspective

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

Breaking news! A Russian billionaire has bought a yet undisclosed NBA team. He's planning to invest huge effort and money into making that team the best. And in fact he's been very specific about the expected result: the first place.

Being his advisor, you need to determine whether it's possible for your team to finish first in its division or not.

More formally, the NBA regular season is organized as follows: all teams play some games, in each game one team wins and one team loses. Teams are grouped into divisions, some games are between the teams in the same division, and some are between the teams in different divisions.

Given the current score and the total number of remaining games for each team of your division, and the number of remaining games between each pair of teams in your division, determine if it's possible for your team to score at least as much wins as any other team in your division.

## Формат входного файла

The first line of the input file contains  $N$  ( $2 \leq N \leq 20$ ) — the number of teams in your division. They are numbered from 1 to  $N$ , your team has number 1.

The second line of the input file contains  $N$  integers  $w_1, w_2, \dots, w_N$ , where  $w_i$  is the total number of games that  $i^{th}$  team has won to the moment.

The third line of the input file contains  $N$  integers  $r_1, r_2, \dots, r_N$ , where  $r_i$  is the total number of remaining games for the  $i^{th}$  team (including the games inside the division).

The next  $N$  lines contain  $N$  integers each. The  $j^{th}$  integer in the  $i^{th}$  line of those contains  $a_{ij}$  — the number of games remaining between teams  $i$  and  $j$ . It is always true that  $a_{ij} = a_{ji}$  and  $a_{ii} = 0$ , for all  $i$   $\sum_j a_{ij} \leq r_i$ .

All the numbers in the input file are non-negative and don't exceed 10 000.

## Формат выходного файла

On the only line of output, print “YES” (without quotes) if it's possible for the team 1 to score at least as much wins as any other team of its division, and “NO” (without quotes) otherwise.

## Пример

<b>i.in</b>	<b>i.out</b>
3 1 2 2 1 1 1 0 0 0 0 0 0 0 0 0	YES
3 1 2 2 1 1 1 0 0 0 0 0 1 0 1 0	NO

## Разбор задачи I. Perspective

Ясно, что можно считать, что все оставшиеся матчи наша команда выигрывает.

Построим граф: исток, из которого дуги пропускной способности 1 идут к вершинам, соответствующим матчам, от каждого из которых есть две

дуги пропускной способности 1 в участвующие команды, а от каждой команды в сток ведет дуга такой пропускной способности, сколько очков может набрать эта команда, не обогнав нас.

Тогда, если максимальный поток в этом графе насыщает дуги из истока, мы можем победить.

## Задача J. Enjoy Arithmetic Progressions

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

Young Andrew loves arithmetic progressions. His even younger sister Anya has written a sequence of numbers on a blackboard. Now Andrew wants to split this sequence into several arithmetic progressions. For example, a sequence 1 2 5 7 9 11 3 can be split into three arithmetic progressions: 1 2, 5 7 9 11 and 3. There is another way to split it into three arithmetic progressions (1 2, 5 7 9 and 11 3), but there is no way to split it into two or less progressions.

Obviously, Andrew would love to split the sequence into as little progressions as possible. He's even willing to change some of the numbers so that the resulting number of arithmetic progressions is smaller. But it would be boring to just change all numbers to 1 2 3 ... .

So Andrew has decided to assign a score of  $c$  to each change operation, and a score of  $p$  to each resulting arithmetic progression. If he changes  $n_c$  numbers and splits the result into  $n_p$  progressions, his total score is  $cn_c + pn_p$ . He wants his total score to be as small as possible.

### Формат входного файла

The first line of the input file contains three integers  $n$ ,  $1 \leq n \leq 3\,000$  (the length of Anya's sequence),  $c$  and  $p$ ,  $1 \leq c, p \leq 10\,000$ . The second line of the input file contains  $n$  integers between  $-1000$  and  $1000$ , inclusive — Anya's sequence.

### Формат выходного файла

In the first line of the output file print minimal total score. In the second line print the number of arithmetic progressions that Andrew will form. Then output the progressions themselves, one per line. Each line should start with the number of numbers in the progression, followed by the numbers themselves. Every number should be written either as integer between  $-10^9$

and  $10^9$ , inclusive, or as a rational number with numerator between  $-10^9$  and  $10^9$ , inclusive, and denominator between 2 and  $10^9$ , inclusive, with the greatest common divisor of numerator and denominator equal to 1. Andrew never uses numbers that can't be written under above restrictions.

## Примеры

j.in	j.out
11 2 5	19
-100 -100 -100 1 1 2	3
2 3 100 100 100	3 -100 -100 -100
	5 1 3/2 2 5/2 3
	3 100 100 100

## Разбор задачи J. Enjoy Arithmetic Progressions

Задача решается методом динамического программирования.

Первая идея — в каждой полученной прогрессии длины больше 1 хотя бы два числа останутся неизменными (так как все числа кроме двух всегда можно переделать так, чтобы получилась прогрессия).

Вторая идея — можно считать, что в каждой полученной прогрессии последнее число останется неизменным. Действительно, если мы меняем несколько последних чисел в прогрессии, то можно их перенести в следующую прогрессию. Единственное исключение — последняя прогрессия, и это нужно отдельно обработать (когда будем считать окончательный ответ, нужно к последней прогрессии уметь добавить все оставшиеся числа, считая их всех измененными).

Наконец, решение: будем считать оптимальное разбиение первых  $k$  чисел на прогрессии, в каждой из которых последнее число неизменно и еще хотя бы одно число неизменно (если только прогрессия не длины 1). Для этого перебираем размер последней прогрессии, и для каждого размера нам нужно выбрать разность прогрессии оптимальным образом. Но это просто: каждое число, кроме последнего, дает нам ровно одну разность прогрессии, при котором это число останется неизменным. Ясно, что нам нужно выбрать ту разность, которая повторяется наибольшее число раз. Это можно сделать быстро, используя структуру данных с операциями “добавить число” и “узнать число, добавленное больше всего раз, и соответствующее количество раз”. Такой структурой данных является, например, хеш-таблица, вида: число → количество раз, плюс одна переменная для хранения текущего максимума.

Время работы  $O(N^2)$ .

## Задача K. Completely Non-zero Determinant

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

Given an integer  $n$ , you need to construct a  $n \times n$  matrix  $M$  of zeroes and ones such that for every  $m$ ,  $1 \leq m \leq n$ ,  $a$ ,  $1 \leq a \leq n - m + 1$  the sub-matrix formed by rows 1 through  $m$  and columns  $a$  through  $a + m - 1$  of  $M$  is non-singular over  $\mathbb{F}_2$ .

Let us remind you that a  $m \times m$  matrix  $P$  over  $\mathbb{F}_2$  is non-singular when there's an odd number of permutations  $p$  of  $1, 2, \dots, m$  such that elements  $P_{1,p_1}, P_{2,p_2}, \dots, P_{m,p_m}$  are all equal to one.

### Формат входного файла

The first and only line of the input file contains an integer  $n$ ,  $1 \leq n \leq 100$ .

### Формат выходного файла

Output the required matrix in  $n$  lines of  $n$  integers (zeroes or ones) each, separated with single spaces inside a line.

### Примеры

<b>k.in</b>	<b>k.out</b>
3	1 1 1 1 0 1 1 0 0

## Разбор задачи K. Completely Non-zero Determinant

Задачу можно решать двумя способами: прямым построением ответа и методом Гаусса.

Прямое построение ответа:  $a_{ij} = C_j^i$ .

Методом Гаусса: запустив метод Гаусса для всей матрицы, для всей матрицы без первого столбца, и так далее, мы можем последовательно вычислять все элементы верхнего треугольника матрицы (элементы нижнего треугольника можно выбрать произвольно).

## Задача L. Fast Typing

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

Vasya has little experience in typing, thus he has to look at the keyboard to locate the necessary keys, and still makes typos while doing so. For simplicity sake, we'll assume that the only type of typo he makes is replacing a character by another one. To correct those, he employs the following strategy: from time to time, he looks at the screen, and if there is any typo in the text, he removes all the characters from the end of the text to the first typo he made inclusive (i.e., he leaves only the correct part of the text intact) by pressing ‘backspace’ key several times, and continues typing from that position again.

Pressing any key (including ‘backspace’) takes 1 unit of time, and looking at the screen takes  $t$  units of time. Given the probabilities of making an error for each character in the text, compute the minimal possible expected time to type the entire text correctly (including verifying that by looking at the screen in the end and noticing no typos).

The text is  $n$  characters long, and the probability of mistyping  $i$ -th character is equal to  $a_i$ .

### Формат входного файла

The input file contains two integer numbers  $n$  and  $t$  ( $1 \leq n \leq 3000$ ,  $1 \leq t \leq 10^6$ ), followed by  $n$  real numbers  $a_i$  ( $10^{-5} \leq a_i \leq \frac{1}{2}$ ).

### Формат выходного файла

Output one real number — the minimal possible expected time. Your answer will be considered correct if it is within  $10^{-6}$  relative error of the exact answer.

### Пример

<b>l.in</b>	<b>l.out</b>
3 1	8.000080000800008
0.00001 0.5 0.00001	

### Разбор задачи L. Fast Typing

Задача решается стандартным динамическим программированием: найдем минимальное ожидаемое время, чтобы набрать оставшийся текст, если мы уже набрали и проверили первые  $k$  символов.

## Задача М. XYZX 2009

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

The annual XYZX programming competition is well known for its unusual problem statements. The Technical Coordinator of the competition, also known as Snusmumrik, intentionally leaves some freedom in the problem statements, so that the participants have to guess the missing conditions from the sample tests, from the problem preface or even epigraph, or just invent something themselves. This peculiarity revealed itself in the latest XYZX-2009 competition as well.

Misha is competing in a round of that competition — Quarterfinal of Eniseysk, and he has bumped into that obstacle again: he needs to understand why does his perfectly correct program fail on the sample tests. The statements are in English, and he knows that there is only one false sentence in the whole problem statement. Moreover, he knows a set of rules that enables him to restore the original statement that the author had in mind. It is very simple: the word “not” can be inserted after any of the words “can”, “may”, “must”, “should” in the text; also the word “no” can be inserted after any of the phrases “is”, “are”. Words “not” and “no” must be inserted exactly as written: “Not”, “nO” and other variants are not permitted. The other words mentioned, however, are case insensitive, so you can insert “not” after “Can”, and you can insert “no” after “ARE”. If there is any article (“a” or “the”) after “is” or “are” where you’re inserting “no”, you must also remove that article.

Misha wants to get the list of all the possible reasons that his program doesn’t work on the sample tests. A reason here is a sentence where a negation can be applied. Remember that no two negations can be applied simultaneously, neither in one sentence nor in different sentences.

### Формат входного файла

The input file contains the text of the problem statement containing latin letters, single spaces and dots. Every sentence has a dot at the end (which is not considered the part of the last word). Words are separated with a single space. There is no space before any dot. There is exactly one space after each dot unless that dot follows the last sentence; in that case, the dot is followed by a newline, and then the input file ends. When inserting a word “not” or “no” after some word, you must first insert a single space, then the corresponding word immediately. The size of the input file doesn’t exceed 1 024 bytes.

## **Формат выходного файла**

The output file must first contain an integer  $k$  on a line by itself — the number of ways the problem statement can be understood differently. Then  $k$  lines must follow. Each line should contain one sentence from the initial text with exactly one negation applied. You must list all the possibilities (the lines are wrapped in the sample output just for better visibility; you must output one sentence per line). All the variants of negation must go in the natural order: all the negations that can be inserted in the first sentence, then all the negations that can be inserted in the second sentence, etc. Inside one sentence, first goes the negation that is applied after the first word (if applicable), then the one that is applied after the second word, etc.

You must not change the case of the letters in the words that you copy from the initial sentences. Your words must still be separated with single spaces, and the same rules apply to the dots at the ends of the sentences, as they do for the input sentences.

*See next page for sample input and output.*

## Пример

### **m.in**

There is a field with label K on the field.  
There is a field with label G on the field.  
You must intersect the rectangle with each  
of the given rectangles separately. Two  
different particles can annihilate each  
other in case of collision. There must be a  
blank line after each test case output. Two  
different ways to insert a symbol into the  
expression are considered the same if the  
resulting expressions viewed as strings are  
equal. Misha can find out whether this  
problem statement can be understood  
correctly at all.

### **m.out**

9

There is no field with label K on the field.  
There is no field with label G on the field.  
You must not intersect the rectangle with  
each of the given rectangles separately.  
Two different particles can not annihilate  
each other in case of collision.  
There must not be a blank line after each  
test case output.  
Two different ways to insert a symbol into  
the expression are no considered the same if  
the resulting expressions viewed as strings  
are equal.  
Two different ways to insert a symbol into  
the expression are considered the same if  
the resulting expressions viewed as strings  
are no equal.  
Misha can not find out whether this problem  
statement can be understood correctly at  
all.  
Misha can find out whether this problem  
statement can not be understood correctly at  
all.

## Разбор задачи М. XYZХ 2009

В задаче просто нужно аккуратно реализовать то, что написано в условии.

### Задача N. Greatest Greatest Common Divisor

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

Andrew has just made a breakthrough in sociology: he realized how to predict whether two persons will be good friends or not. It turns out that each person has an inner *friendship number* (a positive integer). And the *quality of friendship* between two persons is equal to the greatest common divisor of their friendship number.

That means there are *prime* people (with a prime friendship number) who just can't find a good friend, and... Wait, this is irrelevant to this problem.

You are given a list of friendship numbers for several people. Find the highest possible quality of friendship among all pairs of given people.

#### Формат входного файла

The first line of the input file contains an integer  $n$  ( $2 \leq n \leq 100\,000$ ) — the number of people to process. The next  $n$  lines contain one integer each, between 1 and 1 000 000 (inclusive), the friendship numbers of the given people. All given friendship numbers are distinct.

#### Формат выходного файла

Output one integer — the highest possible quality of friendship. In other words, output the greatest greatest common divisor among all pairs of given friendship numbers.

#### Пример

n.in	n.out
4	
9	
15	
25	
16	

## Разбор задачи N. Greatest Greatest Common Divisor

Задача решается очень просто — нужно просто для каждого числа выписать все делители и найти тот, который встречается хотя бы два раза.

### Самое короткое решение: Distinct Substrings

Имя входного файла:	<code>distinct.in</code>
Имя выходного файла:	<code>distinct.out</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

A well-known application of suffix trees is solving the following problem: given a string, find the number of distinct substrings that the string has. For example, the string “abac” has 9 distinct substrings: “a”, “b”, “c”, “ab”, “ba”, “ac”, “aba”, “bac”, “abac”.

You are faced with generating testcases for this problem.

More specifically, you should find the shortest string consisting only of lowercase English letters that has exactly the given amount  $n$  of distinct substrings. Among several shortest strings, choose the lexicographically smallest one.

#### Формат входного файла

First and only line of the input file contains an integer  $n$ ,  $1 \leq n \leq 300$ .

#### Формат выходного файла

In the only line of the output file write the sought string.

#### Примеры

<code>distinct.in</code>	<code>distinct.out</code>
5	aab

### Разбор задачи. Distinct Substrings

Попробуем решить задачу жадно: сначала выберем длину ответа как минимальное число  $l$  такое, что  $\frac{l \cdot (l-1)}{2} \geq n$ . Ясно, что меньшей длиной строки не обойтись.

Будем строить нашу строку слева направо, и каждый раз ставить наименьший возможный символ. Что значит наименьший возможный? Такой, что если после него все остальные буквы будут разные и отличающиеся от всех предыдущих, наберется хотя бы  $n$  различных подстрок.

Ясно, что такое решение всегда находит строку, которая лексикографически меньше либо равнациальному ответу, и имеющую хотя бы  $n$  различных подстрок. Но тогда если она имеет ровно  $n$  различных подстрок, то задача решена!

Дальше мы просто запускаем этот алгоритм на всех входных данных и проверяем, решает ли он задачу. Оказывается, что решает для всех  $n$  кроме 4 и 16 — ну а для них можно решить перебором.

## День девятый (20.02.2011 г.) Конкурс Гольдштейна Виталия Борисовича

### Об авторе...

**Гольдштейн Виталий Борисович**, аспирант Московского Физико-Технического Института. Член научного комитета всероссийской олимпиады школьников по информатике и сборов по подготовке к международной олимпиаде школьников. Стажер компании Google (зима-весна 2008). Завуч Летней Компьютерной Школы (август, 2009) и Летней Компьютерной Школы (зима, 2009-2010). Член жюри Московского и Саратовского четвертьфинала ACM ICPC, член жюри полуфинала NEERC ACM ICPC. Разработчик компании Яндекс. Лектор курса “Алгоритмы и структуры данных” базовой кафедры Яндекса “Анализ данных” на факультете Инноваций и высоких технологий в МФТИ. Член тренерской группы проекта “Яндекс-тренировки” в Москве.



#### Основные достижения:

- серебряная медаль чемпионата мира по программированию среди студентов (ACM ICPC Tokyo 2007);
- участник финалов TopCoder Open, TopCoder Collegiate Open, Global Google Code Jam, Google Code Jam Europe;
- тренер призеров (4-е место, золотая медаль) чемпионата мира по спортивному программированию ACM-ICPC World Finals 2009, Швеция, Стокгольм.

## Теоретический материал. Применение обхода в глубину

### Обход в глубину

#### *Описание алгоритма*

Обход в глубину — это рекурсивный алгоритм обхода графа. Самый простой способ описать этот алгоритм — это привести его реализацию:

```
def dfs(v):
    used[v] = True
    for u in incidents[v]:
        if not used[u]: dfs(u)
```

Кратко можно сказать, что этот алгоритм вызывает себя рекурсивно от смежных и еще непосещенных вершин. Несмотря на свою простоту, этот алгоритм имеет огромное количество применений, благодаря порядку, в котором обходятся вершины. Функция обхода в глубину для каждой вершины будет вызвана не более одного раза, поэтому суммарное время работы алгоритма равно сумме степеней всех вершин, то есть  $O(E)$

#### *Цвета вершин в процессе алгоритма*

В процессе алгоритма каждой вершине соответствует цвет. Вершины, которые еще не были посещены алгоритмом, называют *белыми*. Вершины, которые еще обрабатываются алгоритмом (то есть алгоритм зашел в функцию, но еще не закончил), называют *серыми*. Полностью обработанные вершины, называют *черными*.

```
def dfs(v):
    color[v] = GRAY
    for u in incidents[v]:
        if not used[u]: dfs(u)
    color[v] = BLACK
```

В любой момент работы алгоритма серые вершины образуют некоторый путь от стартовой вершины до текущей. В дальнейшем мы будем использовать понятие цвета вершины в рассуждениях

#### *Время начала и окончания обработки вершин*

Для вершин так же можно определить время начала и окончания обработки алгоритмом. Будем считать, что время идет непрерывно вперед в процессе работы алгоритма. Нам не так важны числовые значения этих времен. В дальнейших рассуждениях мы будем использовать только соотношение этих времен.

```
def dfs(v):
    time_in[v] = cur_time++
    for u in incidents[v]:
        if not used[u]: dfs(u)
    time_out[v] = cur_time++
```

С первого взгляда кажется, что алгоритм закончит обработку вершины после того как обработает все достижимые из нее вершины. Однако это не так. Путь до некоторых вершин может быть загорожен серыми вершинами. Однако верна лемма о белом пути. Пусть произошел вызов алгоритма от некоторой вершины  $v$ , тогда все вершины, достижимые из  $v$  по белым вершинам будут обработаны до окончания обработки  $v$ .

### ***Дерево обхода в глубину***

Деревом обхода в глубину называют множество ребер, по которым прошел обход в глубину. То есть те ребра, которые при просмотре их обходом в глубину вели из текущей вершины в белую. Это понятие обычно применяют для неориентированных графов.

### ***Классификация ребер***

Ребра относительно любого оствового дерева в неориентированном графе классифицируют следующим образом:

1. Ребра дерева.
2. Перекрестные ребра — ребра, соединяющие разные поддеревья.
3. Возвратные ребра — ребра, ведущие от вершины к ее предку.

В ориентированном графе можно классифицировать ребра похожим образом.

1. Ребра дерева — из серой в белую вершину.
2. Перекрестные ребра — из серой в черную вершину.
3. Возвратные ребра — из серой в серую вершину.

Стоит обратить внимание, что при обходе в глубину в неориентированном графе перекрестных ребер нет.

## ***Стандартные применения. Быстрый обзор.***

### ***Компоненты связности***

Компонента связности в неориентированном графе — наибольшее по включению множество попарно достижимых вершин. При запуске обхода

в глубину из вершины  $v$  в неориентированном графе будут помечены все вершины компоненты связности.

### **Топологическая сортировка**

Топологическая сортировка ациклического ориентированного графа — это упорядочивание вершин графа так, чтобы ребра были направлены от меньшего номера к большему. Топологическая сортировка эквивалентна сортировке вершин по возрастанию времени окончания обработки.

### **Мосты и точки сочленения**

Мост в неориентированном графе — ребро, при удалении которого увеличивается количество компонент связности. Точка сочленения в неориентированном графе — вершина, при удалении которой увеличивается количество компонент связности.

Определим функцию  $uptime(v)$  как наименьшее время входа вершины, в которую можно попасть из  $v$  путем спуска по дереву обхода в глубину и последующего подъема по одному возвратному ребру. Эту функцию можно вычислять в процессе обхода в глубину:  $uptime(v) = \min(tin[v], \min_{u \in back(v)}(tin[u]), \min_{u \in tree(v)}(uptime[u]))$ , где  $back(v)$  — множество вершин, в которые ведут возвратные ребра из  $v$ , а  $tree(v)$  — множество потомков вершины  $v$  в дереве обхода в глубину.

Ребро  $(u, v)$  является мостом тогда и только тогда, когда оно ребро дерева ( $u$  предок  $v$ ) и  $uptime(v) > tin[u]$ .

Вершина  $v$  является точкой сочленения в одном из двух случаев:

1.  $v$  — корень дерева и у него хотя бы 2 потомка.
2. Существует потомок  $u$  такой, что  $uptime(v) \geq tin[u]$ .

### **Компоненты вершинной и реберной двусвязности**

- Компонентой реберной двусвязности называют наибольшее по включению множество вершин, такое что между любой парой есть два непересекающихся по ребрам пути.
- Компонентой вершинной двусвязности называют наибольшее по включению множество ребер, такое что в порожденном подграфе между любой парой вершин есть два непересекающихся по вершинам пути.

### **Компоненты сильной связности**

В ориентированном графе понятие компоненты связности не определено. Однако абсолютно так же можно сформулировать определение компонент сильной связности.

Компонента сильной связности в ориентированном графе — наибольшее по включению множество попарно достижимых вершин. Для нахождения необходимо запустить обход в глубину из каждой не помеченной вершины. Упорядочить вершины по убыванию времени выхода. После чего запускать обход в глубину из вершин в полученном порядке. Каждый обход пометит вершины из очередной компоненты сильной связности. Кроме этого, компоненты сильной связности будут выписаны в порядке топологической сортировки.

### ***Конденсация графа***

Конденсацией графа  $G$  называют граф  $H$ , в котором некоторое множество вершин графа  $G$  объединяются в одну вершину графа  $H$ . В графе  $H$  вершины соединены ребром, если соединена хотя бы одна пара вершин из соответствующих множеств графа  $G$ .

Конденсация графа обычно лишена некоторого свойства. Так, например, конденсация компонент сильной связности является ациклическим графом. А конденсация компонент реберной двусвязности является деревом.

## **Дополнительные применения**

### ***Нахождение цикла***

Цикл в графе существует тогда и только тогда, когда в процессе обхода в глубину было обнаружено возвратное ребро. Если ребро обнаружено, то цикл найден, а значит он существует. Несколько более сложно заметить, что при наличии цикла, он обязательно будет обнаружен. Рассмотрим произвольный цикл. Пусть, не ограничивая общности, первой мы посетили вершину  $u$ , тогда  $v$  — это предыдущая вершина этого цикла. По лемме о белом пути вершина  $v$  будет посещена до того, как закончит свою обработку вершина  $u$ . А следовательно, ребро  $(v, u)$  будет возвратным.

### ***Отношение предка в дереве***

Задача состоит в том, чтобы при некотором предподсчете за  $O(1)$  проверять является ли одна вершина предком другой в дереве. Запустим обход в глубину из корня дерева и запомним для каждой вершины время входа и время выхода. Совершенно очевидно, что все потомки вершины  $p$  будут удовлетворять следующему свойству:  $tin[p] \leq tin[v] \ \& \ tout[v] \leq tout[p]$ . Времена входа и выхода в обходе в глубину образовывают правильную скобочную последовательность, поэтому достаточно проверить  $tin[p] \leq tin[v] \leq tout[p]$ .

### ***Нахождение вершин и ребер, которые могут быть на пути от $s$ к $t$***

Задача состоит в том, чтобы найти вершины и ребра, которые в принципе могут находиться на некотором (возможно не простом) пути от  $s$  к  $t$ . Для этого применяется стандартный прием поиска с двух концов. Найдем все вершины достижимые из  $s$ , после чего в транспонированном графе  $G^T$  найдем все вершины достижимые из  $t$  (то есть вершины, из которых достижима  $t$  в графе  $G$ ).

- Вершина  $v$  может лежать на пути, если она достижима из  $s$  и из нее достижима  $t$ .
- Ребро  $(u, v)$  может лежать на пути, если  $u$  достижима из  $s$  и из  $v$  достижима  $t$ .

### ***Проверка на сильносвязность***

Задача состоит в том, чтобы проверить, является ли граф сильносвязным. Мы уже умеем выделять компоненты сильной связности, но для проверки на сильносвязность можно применить более простой алгоритм. Выберем произвольную вершину  $v$ . Есть два необходимых условия сильносвязности:

- из  $v$  достижимы все вершины;
- из всех вершин достижима вершина  $v$ .

Оказывается, что этих условий достаточно, так как из любой вершины можно будет добраться в любую другую вершину через  $v$ .

### ***Проверка на “слабо”-связность***

Назовем ориентированный граф “слабо”-связным, если для любой пары вершины  $u$  и  $v$  верно, что либо из  $u$  достижимо  $v$ , либо из  $v$  достижимо  $u$ .

Для решения этой задачи найдем конденсацию графа. Конденсация графа является ациклическим графом. При этом граф слабосвязан тогда и только тогда, когда слабосвязана конденсация. Очевидно, что ациклический ориентированный граф слабосвязан, если он представляет из себя ориентированную цепь.

### ***SAT-2***

Задача SAT-2 — это задача решения уравнения  $\&_{t \in [1, m]} (x_{f_t}^{\alpha_t} \vee x_{s_t}^{\beta_t}) = 1$  в  $B_2$ . В этом уравнение  $n$  переменных.

Для решения этой задачи в первую очередь преобразуем формулу в другой вид  $\&_{t \in [1, m]} (\neg x_{f_t}^{\alpha_t} \rightarrow x_{s_t}^{\beta_t}) = 1$ . А так же добавим симметричное утверждение  $\&_{t \in [1, m]} (\neg x_{s_t}^{\beta_t} \rightarrow x_{f_t}^{\alpha_t}) = 1$ .

В этом виде очевидно, что нам необходимо выполнить  $2m$  следствий. Построим граф из  $2n$  вершин и  $2m$  ребер. Каждой переменной будет соответствовать две вершины  $x_i$  и  $\neg x_i$ . Соединим ребром вершины, которые присутствуют в уравнении в одном следствии.

Теперь наша задача состоит в том, чтобы расставить около вершин 0 и 1 так, чтобы из 1 не было достижимо 0. Кроме этого,  $x_i$  и  $\neg x_i$  должны соответствовать противоположные значения.

Выделим в полученном графе компоненты сильной связности. Каждая компонента должна быть отмечена одним числом. Если для некоторого  $i$   $x_i$  и  $\neg x_i$  попали в одну компоненту, то решения не существует. В противном случае все компоненты сильной связности распадутся на пары симметричных, в которых все переменные противоположны.

Будем двигаться в порядке, противоположном топологической сортировке. Если очередная компонента не отмечена числом, то отметим ее 1, а симметричную 0. Это можно делать, так как симметричная компонента не будет достижима только из отмеченных 0 компонент.

## Задачи и разборы

### Задача А. Цветные волшебники – 2

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

Сказочная страна представляет собой множество городов, соединенных дорогами с двухсторонним движением. Причем из любого города страны можно добраться в любой другой город либо непосредственно, либо через другие города. Известно, что в сказочной стране не существует дорог, соединяющих город сам с собой и между любыми двумя разными городами, существует не более одной дороги.

В сказочной стране живут желтый и синий волшебники. Желтый волшебник, пройдя по дороге, перекрашивает ее в желтый цвет, синий — в синий. Как известно, при наложении желтой краски на синюю, либо синей краски на желтую, краски смешиваются и превращаются в краску зеленого цвета, который является самым нелюбимым цветом обоих волшебников.

В этом году в столице страны (городе  $f$ ) проводится конференция волшебников. Поэтому желтый и синий волшебники хотят узнать, какое минимальное количество дорог им придется перекрасить в зеленый цвет, чтобы добраться в столицу. Изначально все дороги не покрашены.

Начальное положение желтого и синего волшебников заранее не известно. Поэтому необходимо решить данную задачу для  $k$  возможных случаев их начальных расположений.

### Формат входного файла

Первая строка входного файла содержит целые числа:  $n$  ( $1 \leq n \leq 100\,000$ ) и  $m$  ( $1 \leq m \leq 500\,000$ ) — количество городов и дорог в волшебной стране соответственно. Третья строка содержит одно целое число  $f$  ( $1 \leq f \leq n$ ) — номер города, являющегося столицей сказочной страны. В следующих  $m$  строках находится описание дорог страны. В этих  $m$  строках записано по два целых числа  $a_i$  и  $b_i$ , означающих, что существует дорога, соединяющая города  $a_i$  и  $b_i$ . Следующая строка содержит целое число  $k$  ( $1 \leq k \leq 100\,000$ ) — количество возможных начальных расположений волшебников. Далее следуют  $k$  строк, каждая из которых содержит два целых числа — номера городов, в которых изначально находится желтый и синий волшебники соответственно.

### Формат выходного файла

Для каждого из  $k$  случаев ваша программа должна вывести в выходной минимальное количество дорог, которое придется покрасить в зеленый цвет волшебникам для того, чтобы добраться в столицу.

### Пример

<b>a.in</b>	<b>a.out</b>
6 6	1
1	2
1 2	
2 3	
3 4	
4 2	
4 5	
3 6	
2	
5 6	
6 6	

## Разбор задачи А. Цветные волшебники – 2

Для решения задачи найдем конденсацию графа по компонентам реберной двусвязности. В любой компоненте двусвязности два волшебника с легкостью смогут разойтись, не пересекаясь по ребрам. Мосты — единственные ребра, которые они с неизбежностью должны вместе пройти. Конденсация является деревом, состоящим только из мостов. Требуемое количество ребер — это количество общих ребер в конденсации на пути к столице. Для нахождения количества общих ребер в пути, можно подвесить дерево за компоненту двусвязности со столицей. После этого количество общих ребер будет равно глубине наименьшего общего предка. Нужно использовать любой алгоритм нахождения минимального общего предка за  $O(\log N)$ .

## Задача В. Граф операций

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

Вам задан неориентированный граф. Каждому ребру  $(u, v)$  в этом графе приписана некоторая бинарная операция и некоторое число  $c$ . Число  $c$  может иметь значение 0 или 1, а операции могут быть такие: логическое умножение (AND), логическое сложение (OR) и сложение по модулю два (XOR). Ваша задача выяснить, можно ли присвоить каждой вершине  $u$  число 0 или 1 (обозначим это значение как  $A(u)$ ) таким образом, чтобы для каждого ребра  $(u, v)$  результат приписанной ему бинарной операции над величинами  $A(u)$  и  $A(v)$  был равен написанному на нем числу  $c$ .

Правила вычисления указанных операций такие:

- $(0 \text{ AND } 1) = (1 \text{ AND } 0) = (0 \text{ AND } 0) = 0, (1 \text{ AND } 1) = 1$
- $(0 \text{ OR } 1) = (1 \text{ OR } 0) = (1 \text{ OR } 1) = 1, (0 \text{ OR } 0) = 0$
- $(0 \text{ XOR } 1) = (1 \text{ XOR } 0) = 1, (1 \text{ XOR } 1) = (0 \text{ XOR } 0) = 0$

## Формат входного файла

В первой строке записано два целых числа  $n$  и  $m$  ( $1 \leq n \leq 1000$ ,  $0 \leq m \leq 300000$ ) — количество вершин и ребер в графе. Следующие  $m$  строк содержат описание ребер. Ребро задается номерами соединяемых

вершин  $u_i, v_i$  ( $1 \leq u_i, v_i \leq n, u_i \neq v_i$ ), числом  $c_i$  и названием операции (AND, OR или XOR). Никакое ребро не дано более одного раза.

### Формат выходного файла

Выведите “YES”, если можно найти требуемый набор значений для вершин, и “NO” — в противном случае.

### Примеры

b.in	b.out
3 3 1 2 1 OR 2 3 1 AND 1 3 1 XOR	YES
3 2 1 2 1 AND 2 3 0 OR	NO

### Разбор задачи В. Граф операций

В задаче каждое ребро представляет собой некоторое уравнение из двух переменных. Любое логическое выражение из двух переменных можно привести к нормальной дизъюнктивной форме. После такого приведения задача сводится к задаче SAT-2. Описание решения задачи SAT-2 смотрите в лекции.

### Задача С. Регулярное паросочетание

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

Максим с детства увлекается теорией графов. Сейчас его интересует все, что связано с двудольными графами. Двудольным графом называют граф, вершины которого можно разбить на две части так, что не существует ребер, соединяющих вершины из одной части. В одной замечательной книжке Максим прочитал, что в регулярном двудольном графе существует совершенное паросочетание. В этой же книге было написано, что регулярный граф — это граф, у которого степени всех вершин одинаковые, а совершенное паросочетание — разбиение **всех** вершин графа на пары соединенных ребром вершин. Однако, в этой книге ничего не было написано

как найти это паросочетание. Всю ночь Максим провел в поиске алгоритма, однако нашел только маленькую сноска в огромной книге, что эта задача легко решается для двудольных регулярных графов с степенью  $2^k$ . Помогите ему найти совершенное паросочетание.

### Формат входного файла

В первой строке задано число  $n$  ( $1 \leq n \leq 50000$ ) — количество вершин в каждой доле графа. Во второй строке записана степень каждой вершины  $d$  ( $d = 2^k$ , где  $0 \leq k \leq 5$ ). В последующих  $n$  строках для каждой вершины первой доли записано по  $d$  чисел — номера смежных вершин второй доли. Вершины первой и второй доли нумеруются независимо от 1 до  $n$ . В графе могут быть кратные ребра. Гарантируется, что степени всех вершин, как первой, так и второй доли, равны  $d$ . В графе допускаются кратные ребра, то есть пару вершин могут соединять более одного ребра.

### Формат выходного файла

Выведите ровно  $n$  чисел — для каждой вершины первой доли номер вершины второй доли, с которой она будет объединена в совершенном паросочетании. В выводе должна быть некоторая перестановка чисел от 1 до  $n$ . Если решений несколько, выведите любое.

### Примеры

c.in	c.out
2 1 1 2	1 2
2 2 2 1 1 2	1 2
7 4 5 4 2 6 3 7 5 7 1 3 1 6 2 3 5 1 3 7 6 1 4 6 2 4 2 7 4 5	6 5 3 1 7 2 4

## Разбор задачи С. Регулярное паросочетание

Задача имеет неожиданное отношение к эйлерову циклу. Степень каждой вершины равна степени двойки  $2^d$ . Если  $d = 0$ , то заданный граф и является паросочетанием. Пусть  $d > 0$ , тогда задачу можно свести к  $d - 1$  следующим образом: найдем эйлеров цикл в графе, после чего каждое второе ребро удалим из графа. Полученный граф так же будет являться двудольным регулярным графом, степень каждой вершины которого является степенью двойки. Таким образом за  $d$  шагов мы придем к регулярному графу со степенями 1. Если первоначальное количество ребер  $E$ , то суммарное время работы  $O(E)$ ,  $E + \frac{E}{2} + \frac{E}{4} + \dots + \frac{E}{2^d} \leq 2E$ .

## Задача D. Авиаперелеты

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

Главного конструктора Петю попросили разработать новую модель самолета для компании “Air Бубундия”. Оказалось, что самая сложная часть заключается в подборе оптимального размера топливного бака.

Главный картограф “Air Бубундия” Вася составил подробную карту Бубундии. На этой карте он отметил расход топлива для перелета между каждой парой городов.

Петя хочет сделать размер бака минимально возможным, для которого самолет сможет долететь от любого города в любой другой (возможно, с дозаправками в пути).

### Формат входного файла

Первая строка входного файла содержит натуральное число  $n$  ( $1 \leq n \leq 1000$ ) — число городов в Бубундии. Далее идут  $n$  строк по  $n$  чисел каждая.  $j$ -ое число в  $i$ -ой строке равно расходу топлива при перелете из  $i$ -ого города в  $j$ -ый. Все числа не меньше нуля и меньше  $10^9$ . Гарантируется, что для любого  $i$  в  $i$ -ой строчке  $i$ -ое число равно нулю.

### Формат выходного файла

Первая строка выходного файла должна содержать одно число — оптимальный размер бака.

## Пример

<b>d.in</b>	<b>d.out</b>
4 0 10 12 16 11 0 8 9 10 13 0 22 13 10 17 0	10

## Разбор задачи D. Авиаперелеты

Задача “Авиаперелеты” — это пример задачи на применение двух стандартных алгоритмов. Представьте, что вы контролирующая организация и хотите проверить, можно ли при фиксированном размере бака добраться между любыми двумя городами. Рассмотрим граф, в котором города будут соединены ребром, когда бака хватает, чтобы совершить прямой рейс между ними. Тогда размер бака подходящий, если граф сильносвязан. В лекции описано, как проверить граф на сильносвязность. Теперь можно воспользоваться бинарным поиском по ответу. Выбрав некоторый размер бака, проверим подходит ли такой размер бака. Если размер подходит, то будем искать ответ с меньшим размером бака. Если же граф окажется не сильносвязным, то будем искать ответ с большим баком.

## Задача E. Противопожарная безопасность

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

В городе Зеленоград  $n$  домов. Некоторые из них соединены дорогами с односторонним движением.

В последнее время в Зеленограде участились случаи пожаров. В связи с этим жители решили построить в городе несколько пожарных станций. Но возникла проблема — едущая по вызову пожарная машина, конечно, может игнорировать направление движения текущей дороги, однако, возвращающаяся с задания машина обязана следовать правилам дорожного движения (жители Зеленограда свято чтут эти правила!).

Ясно, что где бы ни оказалась пожарная машина, у неё должна быть возможность вернуться на ту пожарную станцию, с которой выехала. Но строительство станций стоит больших денег, поэтому на совете города

было решено построить минимальное количество станций таким образом, чтобы это условие выполнялось. Кроме того, для экономии было решено строить станции в виде пристроек к уже существующим домам.

Ваша задача — написать программу, рассчитывающую оптимальное положение станций.

### **Формат входного файла**

В первой строке входного файла задано число  $n$  ( $1 \leq n \leq 3\,000$ ). Во второй строке записано количество дорог  $m$  ( $1 \leq m \leq 100\,000$ ). Далее следует описание дорог в формате  $a_i \ b_i$ , означающее, что по  $i$ -й дороге разрешается движение автотранспорта от дома  $a_i$  к дому  $b_i$  ( $1 \leq a_i, b_i \leq n$ ).

### **Формат выходного файла**

В первой строке выведите минимальное количество пожарных станций  $K$ , которые необходимо построить. Во второй строке выведите  $K$  чисел в произвольном порядке — дома, к которым необходимо пристроить станции. Если оптимальных решений несколько, выведите любое.

### **Пример**

<b>e.in</b>	<b>e.out</b>
5	2
7	4 5
1 2	
2 3	
3 1	
2 1	
2 3	
3 4	
2 5	

### **Разбор задачи Е. Противопожарная безопасность**

Требуется найти множество вершин, которые достижимы из всех остальных. Для этого нужно найти конденсацию графа. Из каждой компоненты сильной связности, из которой не выходит ни одного ребра, нужно выбрать по одной вершине. Тогда из любой вершины перемещаясь по произвольному ребру конденсации мы рано или поздно приедем в такую компоненту, из которой не выходит ребер. Минимальность объясняется тем, что из вершин выбранных компонент не достижимы другие вершины.

## Задача F. Островные государства

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

Суровые феодальные времена переживала некогда великая островная страна Байтландия. За главенство над всем островом борются два самых сильных барона. Таким образом, каждый город страны контролируется одним из правителей. Как водится издревле, некоторые из городов соединены двусторонними дорогами. Бароны очень не любят друг друга и стараются делать как можно больше пакостей. В частности, теперь для того, чтобы пройти по дороге, соединяющей города различных правителей, надо заплатить пошлину — один байтландский рубль. Кроме этого, за выезд из городов с четными номерами берется удвоенная пошлина.

Программист Вася живет в городе номер 1. С наступлением лета он собирается съездить в город  $N$  на Всебайтландское собрание программистов. Разумеется, он хочет затратить при этом как можно меньше денег и помочь ему здесь, как обычно, предлагается Вам.

### Формат входного файла

В первой строке входного файла записано два числа  $N$  и  $M$  ( $1 \leq N, M \leq 100\,000$ ) — количество городов и количество дорог соответственно.

В следующий строке содержится информация о городах —  $N$  чисел 1 или 2 — какому из баронов принадлежит соответствующий город.

В последних  $M$  строках записаны пары  $1 \leq a, b \leq N$ ,  $a \neq b$ . Каждая пара означает наличие дороги из города  $a$  в город  $b$ . По дорогам Байтландии можно двигаться в любом направлении.

### Формат выходного файла

Если искомого пути не существует, выведите единственное слово `impossible`. В противном случае, в первой строке напишите минимальную стоимость и количество посещенных городов, а во вторую выведите эти города в порядке посещения. Если минимальных путей несколько, выведите любой.

## Пример

<b>f.in</b>	<b>f.out</b>
7 8 1 1 1 1 2 2 1 1 2 2 5 2 3 5 4 4 3 4 7 1 6 6 7	0 5 1 2 3 4 7
5 5 1 1 1 2 1 1 2 2 3 3 4 4 5 2 4	3 5 1 2 3 4 5

## Разбор задачи F. Островные государства – 2

Требуется найти кратчайший путь в графе, веса ребер которого 0, 1 или 2. Для этого можно воспользоваться модифицированным обходом в ширину. В середину каждого ребра веса 2 поставим фиктивную вершину. Тем самым одно ребро веса 2 превратится в два ребра веса 1. Тем самым мы свели задачу к поиску кратчайшего пути в 0-1 графе. Самым простым способом решить эту задачу — при каждом добавлении вершины в очередь обхода в ширину, запускать обход в глубину по 0 ребрам графа. Все новые вершины, которые мы смогли посетить, добавлять в очередь. Естественно каждую вершину должен посетить либо обход в глубину, либо обход в ширину, причем ровно один раз.

## Задача G. Король

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

В Тридесятом царстве, Тридевятом государстве жил-был король. И было у короля  $n$  сыновей. В Тридесятом царстве жили  $n$  прекрасных девушек, и король знал, какие девушки нравятся каждому сыну (поскольку сыновья были молодыми и безшабашными, то им могли нравиться несколько девушек одновременно).

Однажды король приказал своему советнику подобрать для каждого сына прекрасную девушку, на которой тот сможет жениться. Советник выполнил приказ и подобрал для каждого сына для женитьбы прекрасную девушку, которая ему нравилась. Разумеется, каждая девушка может выйти замуж только за одного из сыновей.

Посмотрев на список невест, король сказал: “Мне нравится этот список, но я хочу знать для каждого сына список всех девушек, на которых он может жениться. Разумеется, при этом все сыновья также должны иметь возможность жениться на девушках, которые им нравятся”.

Эта задача оказалась для советника слишком сложной. Помогите ему избежать казни, решив ее.

### Формат входного файла

Первая строка входного файла содержит число  $n$  — количество сыновей ( $1 \leq n \leq 2\,000$ ). Следующие  $n$  строк содержат списки прекрасных девушек, которые нравятся сыновьям. В начале идет  $k_i$  — количество девушек, которые нравятся  $i$ -му сыну. Затем идут  $k_i$  чисел — номера девушек. Сумма  $k_i$  не превышает 200 000.

Последняя строка входного файла содержит список, составленный советником —  $n$  различных чисел от 1 до  $n$ : для каждого сына — номер прекрасной девушки, на которой он может жениться. Гарантируется, что список корректен, то есть каждому сыну нравится выбранная для него девушка.

### Формат выходного файла

Выходной файл должен содержать  $n$  строк. Для каждого сына выведите  $l_i$  — количество различных девушек, на которых он может жениться. После этого выведите  $l_i$  чисел — номера девушек в произвольном порядке.

## Пример

<b>g.in</b>	<b>g.out</b>
4	2 1 2
2 1 2	2 1 2
2 1 2	1 3
2 2 3	1 4
2 3 4	
1 2 3 4	

## Разбор задачи G. Король

Требуется при заданном максимальном паросочетании найти все ребра, которые могут принадлежать какому-либо максимальному паросочетанию. Так как заданное паросочетание максимально, то не существует чередующегося удлиняющего пути. Ориентируем все ребра, входящие в паросочетание, в направлении от первой ко второй доле. А ребра, входящие в паросочетание из второй доли в первую. Рассмотрим произвольное ребро, не входящее в заданное максимальное паросочетание, ведущее из  $u$  первой доли в  $v$  второй доли. Оно может войти в максимальное паросочетание, если мы найдем чередующуюся цепочку от  $v$  к  $u$ . То есть фактически, ребро  $(u, v)$  находится в некотором цикле. Иначе говоря,  $u$  и  $v$  находятся в одной компоненте сильной связности. Таким образом, нужно выделить компоненты сильной связности. Все ребра, которые соединяют вершины одной компоненты и будут ответом на поставленный вопрос.

## Задача H. Каток

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

Бизнесмен Николай очень любит кататься на коньках. Каждый год 31-го декабря он собирает компанию друзей, арендует каток и весело встречает там Новый Год. Недавно Николаю в голову пришла мысль, что гораздо приятнее было бы иметь собственный каток и бесплатно кататься на нем в любое удобное время. Недолго думая, он решил реализовать эту идею, то есть построить для себя каток, и нанял архитектора Дмитрия.

Архитектор узнал, что любимое число Николая — число  $P$ , а любимый способ катания на коньках — катание вдоль бортика. Чтобы угодить клиенту, Дмитрий решил построить такой каток, на котором Николай, прокатив-

вшись вдоль всех бортиков, проедет ровно  $P$  сотен метров. Ему уже почти удалось нарисовать план такого катка, но Николай неожиданно предъявил новое требование — каток должен иметь площадь ровно  $S$  десятков тысяч квадратных метров (на меньшей площади не поместятся все друзья Николая, а большая слишком дорого обойдется).

Если некоторая площадь катка граничит с площадью, не принадлежащей катку, то между ними обязательно необходимо построить бортик. Таким образом, запрещается устанавливать бортики в произвольном месте катка, а разрешается лишь на его границе.

Новый каток будет построен недалеко от коттеджа Николая. В данный момент под этот проект выделили квадратный участок со стороной 400 метров. Проект катка должен быть таким, чтобы все углы катка были прямые, длины бортиков кратны 100 метрам, а сами бортики параллельны сторонам участка, на котором будет вестись строительство.

Помогите Дмитрию придумать план катка, который удовлетворял бы всем требованиям.

## Формат входного файла

В единственной строке входного файла записаны два целых числа  $P$  и  $S$  ( $1 \leq P \leq 40$ ,  $1 \leq S \leq 16$ ) — периметр и площадь будущего катка, соответственно.

## Формат выходного файла

В выходной файл выведите “Impossible”, если спроектировать каток, удовлетворяющий указанным требованиям невозможно.

В противном случае, выведите 4 строки по 4 символа в каждой — план нового катка. Каждый символ будет описанием квадрата  $100 \times 100$  метров участка, на котором будет вестись строительство. Если соответствующий квадрат участка находится внутри катка, то выведите “\*”, иначе — “.”.

## Примеры

<b>h.in</b>	<b>h.out</b>
18 8	**** *.*. **.. ....
3 1	Impossible

## Разбор задачи Н. Каток

Требуется построить конфигурацию катка в квадрате  $4 \times 4$  с заданной площадью и периметром. Размер квадрата достаточно маленький, поэтому можно перебрать все возможные конфигурации катка, которых  $2^{16}$ . После этого необходимо проверить связность графа и соответствие количества клеток катка и требуемой площади. Для подсчета периметра достаточно посчитать количество полосок сетки, которые соединяют клетку катка и внешности катка. Если периметр будет равен требуемому, то выедем иско-мую конфигурацию.

### Задача I. Предок

Имя входного файла:	<code>i.in</code>
Имя выходного файла:	<code>i.out</code>
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Напишите программу, которая для двух вершин дерева определяет, является ли одна из них предком другой.

### Формат входного файла

Первая строка входного файла содержит натуральное число  $n$  ( $1 \leq n \leq 100\,000$ ) — количество вершин в дереве. Во второй строке находится  $n$  чисел,  $i$ -ое из которых определяет номер непосредственного родителя вершины с номером  $i$ . Если это число равно нулю, то вершина является корнем дерева.

В третьей строке находится число  $m$  ( $1 \leq m \leq 100\,000$ ) — количество запросов. Каждая из следующих  $m$  строк содержит два различных числа  $a$  и  $b$  ( $1 \leq a, b \leq n$ ).

### Формат выходного файла

Для каждого из  $m$  запросов выведите на отдельной строке число 1, если вершина  $a$  является одним из предков вершины  $b$ , и 0 — в противном случае.

## Пример

<b>i.in</b>	<b>i.out</b>
6	0
0 1 1 2 3 3	1
5	1
4 1	0
1 4	0
3 6	
2 6	
6 5	

## Разбор задачи I. Предок

В этой задаче требуется быстро определять, является ли одна вершина предком другой в дереве.

Решение задачи было описано в лекции.

## Задача J. Неизбежность

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

Вася живет в первой вершине связного неориентированного графа, состоящего из  $n$  вершин и  $m$  ребер. Каждый день он ходит в школу, находящуюся в вершине с номером  $n$ . Вася старается каждый деньходить в школу новым маршрутом, однако однажды он заметил, что некоторые ребра он проходит каждый день, независимо от того, каким маршрутом идет. Помогите Васе найти все такие ребра.

## Формат входного файла

Первая строка входного файла содержит два натуральных числа  $n$  и  $m$  — количество вершин и ребер графа соответственно ( $n \leq 20\,000$ ,  $m \leq 200\,000$ ).

Следующие  $m$  строк содержат описание ребер по одному на строке. Ребро номер  $i$  описывается двумя натуральными числами  $b_i$ ,  $e_i$  — номерами концов ребра ( $1 \leq b_i, e_i \leq n$ ).

## Формат выходного файла

Первая строка выходного файла должна содержать одно натуральное число  $b$  — количество ребер, которые неизбежно встречаются на пути Васи. На следующей строке выведите  $b$  целых чисел — номера этих ребер в возрастающем порядке. Ребра нумеруются с единицы в том порядке, в котором они заданы во входном файле.

## Пример

j.in	j.out
7 8 1 2 2 3 3 1 4 3 5 4 5 6 4 6 6 7	2 4 8
4 4 1 2 2 4 1 4 2 3	0

## Разбор задачи J. Неизбежность

Требуется найти ребра, которые гарантированно лежат на пути между двумя заданными вершинами  $s$  и  $t$ . Не трудно понять, что такие ребра — это мосты. Однако не все мосты, а только находящиеся на пути. Для решения задачи необходимо найти мосты. После этого нужно найти произвольный простой путь из  $s$  в  $t$ . Мосты, лежащие на этом пути, и будут решением задачи.

## Задача K. Мосты

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

Дан неориентированный граф. Требуется найти все мосты в нем.

## Формат входного файла

Первая строка входного файла содержит два натуральных числа  $n$  и  $m$  — количество вершин и ребер графа соответственно ( $n \leq 20\,000$ ,  $m \leq 200\,000$ ).

Следующие  $m$  строк содержат описание ребер по одному на строке. Ребро номер  $i$  описывается двумя натуральными числами  $b_i$ ,  $e_i$  — номерами концов ребра ( $1 \leq b_i, e_i \leq n$ ).

## Формат выходного файла

Первая строка выходного файла должна содержать одно натуральное число  $b$  — количество мостов в заданном графе. На следующей строке выведите  $b$  целых чисел — номера ребер, которые являются мостами, в возрастающем порядке. Ребра нумеруются с единицы в том порядке, в котором они заданы во входном файле.

## Пример

<b>k.in</b>	<b>k.out</b>
6 7	1
1 2	3
2 3	
3 4	
1 3	
4 5	
4 6	
5 6	

## Разбор задачи К. Мосты

Нужно просто реализовать алгоритм нахождение мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.

## Задача L. Цветные волшебники

Имя входного файла: 1.in  
Имя выходного файла: 1.out  
Ограничение по времени: 2 с  
Ограничение по памяти: 64 Мб

Сказочная страна представляет собой множество городов, соединенных дорогами с двухсторонним движением. Причем из любого города страны

можно добраться в любой другой город либо непосредственно, либо через другие города. Известно, что в сказочной стране не существует дорог, соединяющих город сам с собой, и между любыми двумя разными городами, существует не более одной дороги.

В сказочной стране живут желтый и синий волшебники. Желтый волшебник, пройдя по дороге, перекрашивает ее в желтый цвет, синий — в синий. Как известно, при наложении желтой краски на синюю, либо синей краски на желтую, краски смешиваются и превращаются в краску зеленого цвета, который является самым нелюбимым цветом обоих волшебников.

В этом году в столице страны (городе  $f$ ) проводится конференция волшебников. Поэтому желтый и синий волшебники хотят узнать, какое минимальное количество дорог им придется покрасить в зеленый цвет, чтобы добраться в столицу. Изначально все дороги не покрашены.

Начальное положение желтого и синего волшебников заранее не известно. Поэтому необходимо решить данную задачу для  $k$  возможных случаев их начальных расположений.

## **Формат входного файла**

Первая строка входного файла содержит целые числа:  $n$  ( $1 \leq n \leq 100$ ) и  $m$  ( $1 \leq m \leq 500$ ) — количество городов и дорог в волшебной стране соответственно. Третья строка содержит одно целое число  $f$  ( $1 \leq f \leq n$ ) — номер города, являющегося столицей сказочной страны. В следующих  $m$  строках, находится описание дорог страны. В этих  $m$  строк записано по два целых числа  $a_i$  и  $b_i$ , означающих, что существует дорога, соединяющая города  $a_i$  и  $b_i$ . Следующая строка содержит целое число  $k$  ( $1 \leq k \leq 100$ ) — количество возможных начальных расположений волшебников. Далее следуют  $k$  строк, каждая из которых содержит два целых числа — номера городов, в которых изначально находится желтый и синий волшебники соответственно.

## **Формат выходного файла**

Для каждого из  $k$  случаев, ваша программа должна вывести в выходной минимальное количество дорог, которое придется покрасить в зеленый цвет волшебникам для того, чтобы добраться в столицу.

## Пример

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

## Разбор задачи L. Цветные волшебники

Для решения задачи найдем конденсацию графа по компонентам реберной двусвязности. В любой компоненте двусвязности два волшебника с легкостью смогут разойтись, не пересекаясь по ребрам. Мосты — единственные ребра, которые они с неизбежностью должны вместе пройти. Конденсация является деревом, состоящее только из мостов. Требуемое количество ребер — это количество общих ребер в конденсации на пути к столице. Для нахождения количества общих ребер в пути можно подвесить дерево за компоненту двусвязности с столицей. После этого количество общих ребер будет равно глубине наименьшего общего предка. Можно искать наименьшего общего предка за линейное время.

## Задача M. Точки сочленения

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

Дан неориентированный граф. Требуется найти все точки сочленения в нем.

## Формат входного файла

Первая строка входного файла содержит два натуральных числа  $n$  и  $m$  — количество вершин и ребер графа соответственно ( $n \leq 20\,000$ ,  $m \leq 200\,000$ ).

Следующие  $m$  строк содержат описание ребер по одному на строке. Ребро номер  $i$  описывается двумя натуральными числами  $b_i, e_i$  — номерами концов ребра ( $1 \leq b_i, e_i \leq n$ ).

### Формат выходного файла

Первая строка выходного файла должна содержать одно натуральное число  $b$  — количество точек сочленения в заданном графе. На следующей строке выведите  $b$  целых чисел — номера вершин, которые являются точками сочленения, в возрастающем порядке.

### Пример

<b>m.in</b>	<b>m.out</b>
9 12	3
1 2	1
2 3	2
4 5	3
2 6	
2 7	
8 9	
1 3	
1 4	
1 5	
6 7	
3 8	
3 9	

### Разбор задачи М. Точки сочленения

Нужно просто реализовать алгоритм нахождение мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.

### Задача N. Почтальон

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

В городе есть  $n$  площадей, соединенных улицами. При этом количество улиц не превышает ста тысяч и существует не более трех площадей, на

которые выходит нечетное число улиц. Для каждой улицы известна ее длина. По улицам разрешено движение в обе стороны. В городе есть хотя бы одна улица. От любой площади до любой можно дойти по улицам.

Почтальону требуется пройти хотя бы один раз по каждой улице так, чтобы длина его пути была наименьшей. Он может начать движение на любой площади и закончить также на любой (в том числе и на начальной).

### Формат входного файла

Первая строка входного файла содержит натуральное число  $n$  — количество площадей в городе ( $1 \leq n \leq 1000$ ). Далее следуют  $n$  строк, задающих улицы. В  $i$ -ой из этих строк находится число  $m_i$  — количество улиц, выходящих из площади  $i$ . Далее следуют  $m_i$  пар положительных чисел. В  $j$ -ой паре первое число — номер площади, в которую идет  $j$ -ая улица с  $i$ -ой площади, а второе число — длина этой улицы.

Между двумя площадями может быть несколько улиц, но не может быть улиц с площади на нее саму.

Все числа во входном файле не превосходят  $10^5$ .

### Формат выходного файла

Если решение существует, то в первую строку выходного файла выведите одно число — количество улиц в искомом маршруте (считая первую и последнюю), а во вторую — номера площадей в порядке их посещения.

Если решений нет, выведите в выходной файл одно число  $-1$ .

Если решений несколько, выведите любое.

### Пример

<b>n.in</b>	<b>n.out</b>
4 2 2 1 2 2 4 1 2 4 4 3 5 1 1 2 2 5 4 8 2 3 8 2 4	5 1 2 3 4 2 1

### Разбор задачи N. Почтальон

Нужно просто реализовать алгоритм нахождение мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.

## Задача О. Конденсация графа

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

Требуется найти количество ребер в конденсации ориентированного графа. Примечание: конденсация графа не содержит кратных ребер.

### Формат входного файла

Первая строка входного файла содержит два натуральных числа  $n$  и  $m$  — количество вершин и ребер графа соответственно ( $n \leq 10\,000$ ,  $m \leq 100\,000$ ). Следующие  $m$  строк содержат описание ребер, по одному на строке. Ребро номер  $i$  описывается двумя натуральными числами  $b_i$ ,  $e_i$  — началом и концом ребра соответственно ( $1 \leq b_i, e_i \leq n$ ). В графе могут присутствовать кратные ребра и петли.

### Формат выходного файла

Первая строка выходного файла должна содержать одно число — количество ребер в конденсации графа.

### Пример

<b>o.in</b>	<b>o.out</b>
4 4	
2 1	
3 2	
2 3	
4 3	2

## Разбор задачи О. Конденсация графа

Нужно просто реализовать алгоритм нахождение мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.

## Задача Р. Топологическая сортировка

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

Дан ориентированный невзвешенный граф. Необходимо его топологически отсортировать.

### Формат входного файла

В первой строке входного файла даны два натуральных числа  $N$  и  $M$  ( $1 \leq N \leq 100\,000, M \leq 100\,000$ ) — количество вершин и рёбер в графе соответственно. Далее в  $M$  строках перечислены рёбра графа. Каждое ребро задаётся парой чисел — номерами начальной и конечной вершин соответственно.

### Формат выходного файла

Вывести любую топологическую сортировку графа в виде последовательности номеров вершин. Если граф невозможно топологически отсортировать, вывести -1.

### Пример

<b>p.in</b>	<b>p.out</b>
6 6	4 6 3 1 2 5
1 2	
3 2	
4 2	
2 5	
6 5	
4 6	
3 3	-1
1 2	
2 3	
3 1	

## Разбор задачи Р. Топологическая сортировка

Нужно просто реализовать алгоритм нахождения мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.