§3. Заключительный этап: индивидуальная часть

3.1. Задачи по математике (9 класс)

Задача 3.1.1 (20 баллов)

Саша едет на велосипеде по ровному участку дороги и хочет измерит свою ско-рость. Он делает 56 оборотов в минуту педалями (легко измерить при помощи часов с секундомером). Еще он знает, что диаметры колес 66 сантиметров, большая звез-дочка имеет 45 зубьев, а маленькая — 16 зубьев. Найдите скорость Саши в км/час. Результат округлите до целого числа.

Решение

Один оборот заднего колеса соответствует одному обороту малой звездочки и $\frac{16}{45}$ оборота большой звездочки. За это время Саша сдвигается на расстояние, равное длине окружности колеса, т.е. 66π см. Угловая скорость заднего колеса равна $\omega = \frac{56 \cdot 45}{16}$ об/мин. Тогда скорость велосипеда равна $v = \frac{56 \cdot 45 \cdot 66\pi}{16}$ см/мин= $6,237\pi$ км/час$\approx$ 20 км/час.

Дополнительные критерии оценки

Отсутствуют.

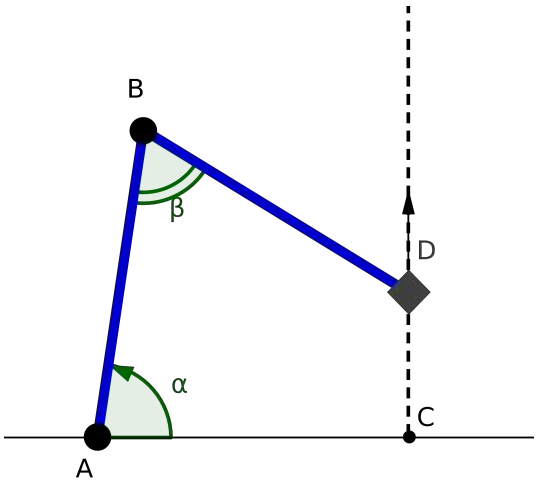
Задача 3.1.2 (30 баллов)

Робот манипулятор состоит из двух стержней длиной 1 м и управляется двумя осями в точках A и B. Предмет D в начальный момент находится в точке C на расстоянии 1 м от A. Требуется поднять его строго вертикально вверх управляя только углами α и β.

а) **(5 балл)** На какую максимальную высоту можно поднять предмет D при помощи этого манипулятора?

б) **(10 балла)** Найдите значения α и β в момент, когда $CD = 0,5$ м.

в) **(15 баллов)** Выразите α и β в виде функций от времени t так, чтобы предмет D поднимался с постоянной скоростью и достиг максимальной высоты в конце первой минуты.



Решение

а) Пусть $h = CD$ высота предмета. По неравенству треугольника $AD \leqslant AB + BD = 2$, причем равенство достигается когда угол $\beta = 180°$. Тогда $h = \sqrt{AD^2 - 1^2} \leqslant \sqrt{3}$.

б) В решение пункта в) вместо $\sqrt{3}t$ поставьте 0,5.

в) Пусть $t \in [0; 1]$ время в минутах, $h(t) = \sqrt{3}t$. По теореме Пифагора $AD^2 = 3t^2 + 1$. Из треугольника ABD по теореме косинусов $AD^2 = AB^2 + BD^2 - 2AB \cdot BD \cos\beta$, откуда $\cos\beta = \frac{1-3t^2}{2}$, $\beta(t) = \arccos(\frac{1-3t^2}{2})$. $\alpha = \angle CAD + \angle DAB$, $\angle CAD = \operatorname{arctg}(\sqrt{3}t)$, $\angle DAB = \arccos\frac{AD}{2} = \arccos\frac{\sqrt{3t^2+1}}{2}$.

https://ggbm.at/wpMJhsXm

Дополнительные критерии оценки

Если пункт в) решен, то за б) тоже начисляются баллы.

Задача 3.1.3 (50 баллов)

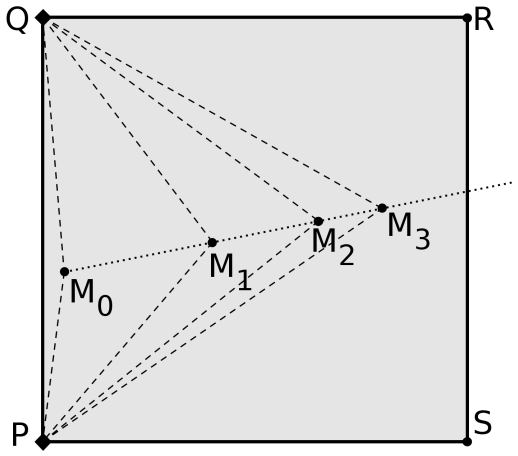
Имеется квадратное поле $PQRS$ 100см×100см. В вершинах P и Q размещены датчики, которые каждую секунду измеряют квадрат расстояния до движущегося объекта в точке M_t. Известно, что объект движется прямолинейно с постоянным ускорением и не меняет направления движения. Обозначим $p_t = PM_t^2$ и $q_t = QM_t^2$. Время t измеряется в секундах. Результаты трех измерений приведены в таблице.

t	0	1	2
p_t	1625	3809	6929
q_t	3625	4409	6529

а) **(10 баллов)** Введите систему координат с началом в точке P, осью абсцисс по направлению луча PS и осью ординат по направлению луча PQ (единица измерения в сантиметрах). Найдите координаты точек M_0, M_1, M_2 в этой системе координат.

б) **(20 баллов)** Какие значения p_3 и q_3 покажут датчики в момент времени $t = 3$с ?

в) **(20 баллов)** Выясните, в какой момент времени остановится объект и покинет ли он поле $PQRS$.



Решение

Для точки $M_t(x_t; y_t)$ выполняются равенства (1) $\begin{cases} x_t^2 + y_t^2 = p_t; \\ (100 - x_t)^2 + y_t^2 = q_t. \end{cases}$

а) Подставляем в систему (1) вместо t значения 0, 1, 2 и, решая систему, находим координаты точек $M_0(5; 40)$, $M_1(40; 47)$, $M_2(65; 52)$.

б) По условию $\begin{cases} x_t = x_0 + v_x t + \frac{a_x}{2} t^2; \\ y_t = y_0 + v_y t + \frac{a_y}{2} t^2. \end{cases}$ Подставляем известные величины при $t = 1$ и $t = 2$, решаем систему линейных уравнений относительно координат скорости и ускорения тела. Получаем зависимость координат тела от времени (2) $\begin{cases} x_t = 5 + 40t - 5t^2; \\ y_t = 40 + 8t - t^2. \end{cases}$ Подстановкой $t = 3$ в (2) и (1) находим $M_3(80; 55)$, $p_3 = 9425$, $q_3 = 8425$.

в) Скорость тела, движущегося равноускоренно по правилам описанным в пункте б), можно описать формулами: $\vec{v}\{40 - 10t; 8 - 2t\}$. Направление движения не менялось, поэтому координаты вектора скорости не меняли знака. В момент $t = 4$ тело остановилось. Его координаты в этот момент $M_4(85; 56)$ — точка внутри поля.

Дополнительные критерии оценки

Отсутствуют.

3.2. Задачи по математике (10-11 класс)

Задача 3.2.1 (20 баллов)

Можно ли многочлен $x^4 + x^3 + 2x^2 + 9x + 15$ представить в виде произведения двух многочленов ненулевой степени с целыми коэффициентами?

Решение

Можно $(x^2-2x+5)(x^2+3x+3)$. Разложение ищем в виде $(x^2+ax+b)(x^2+cx+d)$, где неизвестные коэффициенты a, b, c, d — целые числа. $bd = 15$, перебираем различные варианты (всего 4 случая), для которых коэффициенты a и c находим из условий: $a + c = 1$, $ac + b + d = 2$, $ad + bc = 9$.

Дополнительные критерии оценки

За ответ «Можно» 0 баллов. Пример разложения 20 баллов.

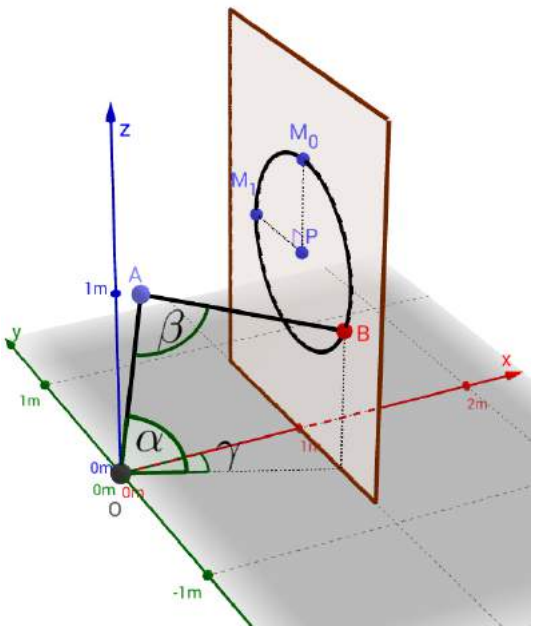
Задача 3.2.2 (30 баллов)

Роботу манипулятору предстоит выполнить плазменную резку в листе стали в виде окружности диаметром 1 м. Тело манипулятора состоит из двух стержней OA и AB длиной 1 м и управляется двумя осями в точке O и еще одной в точке A. Резак находится в конце B. Угол α отвечает за отклонение стержня OA от плоскости Oxy, β — угол между стержнями OA и AB, γ — поворот вокруг оси Oz (отсчитывается от оси Ox против хода часовой стрелки). Лист стали отстоит на расстоянии 1 м от точки O перпендикулярно оси Ox, центр окружности резки в точке $P(1; 0; 1)$. Резка начинается в точке M_0 и совершается против направления часовой стрелки.

а) **(5 балла)** Найдите значения углов α, β и γ в начальный момент резки, т.е. когда резак находится в точке $M_0(1; 0; 1,5)$.

б) **(10 балла)** Найдите значения углов α, β и γ в момент резки, когда резак находится в точке M_1.

в) **(15 баллов)** Выразите α, β и γ в виде функций от времени t так, чтобы манипулятор завершил резку за 4 минуты двигаясь по контуру с постоянной скоростью.



Решение

а) $\alpha = \arccos\dfrac{\sqrt{13}}{4} + \operatorname{arctg} 1,5$; $\beta = \arccos(-0,625)$; $\gamma = 0$.

б) $\alpha(1) = \dfrac{1}{2}\arccos\left(\dfrac{1}{8}\right) + \arcsin\left(\dfrac{2}{3}\right)$, $\beta(1) = \arccos\left(-\dfrac{1}{8}\right)$, $\gamma(1) = \operatorname{arctg}\left(\dfrac{1}{2}\right)$.

в) Зададим точку $M = M(t)$, которая равномерно вращается по окружности резки с периодом 4 минуты. Её можно задать уравнениями

$$M : \begin{cases} x(t) = 1; \\ y(t) = \dfrac{1}{2}\sin\dfrac{\pi t}{2}; \\ z(t) = 1 + \dfrac{1}{2}\cos\dfrac{\pi t}{2}. \end{cases}$$

Будем искать условия на α, β, γ такие, чтобы точка B совпадала с M. По определению $OM^2 = x^2 + y^2 + z^2 = 2,25 + \cos\dfrac{\pi t}{2}$. По теореме косинусов из треугольника AOB получаем условие на β: $OB^2 = 2 - 2\cos\beta = 2,25 + \cos\dfrac{\pi t}{2}$. Откуда $\cos\beta = -\dfrac{1}{8} - \dfrac{1}{2}\cos(\dfrac{\pi t}{2})$. Обозначим N проекцию точки M на плоскость Oxy. Тогда $\angle NOx = \gamma = \operatorname{arctg}\dfrac{y}{x} = \operatorname{arctg}\left(\dfrac{1}{2}\sin(\dfrac{\pi t}{2})\right)$. $\alpha = \angle MON + \angle AOB = \arcsin\dfrac{z}{OM} + \dfrac{1}{2}(180^\circ - \beta) = \arcsin\left(\dfrac{2 + \cos(\frac{\pi t}{2})}{\sqrt{9 + 4\cos(\frac{\pi t}{2})}}\right) + \dfrac{1}{2}\arccos\left(\dfrac{1}{8} + \dfrac{1}{2}\cos(\dfrac{\pi t}{2})\right)$.

Решение задачи по ссылке

https://ggbm.at/zZUZYByv

Дополнительные критерии оценки

Отсутствуют.

Задача 3.2.3 (50 баллов)

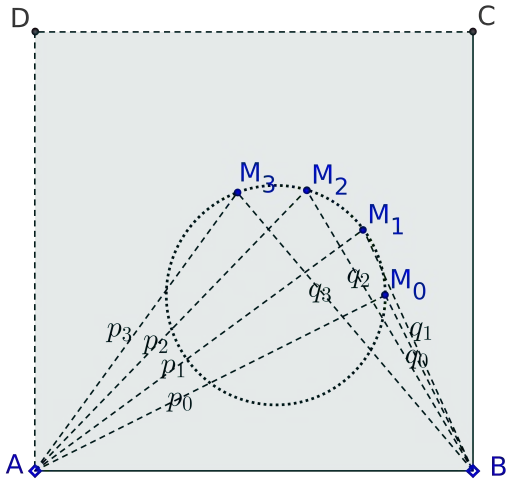
Имеется квадратное поле $ABCD$ 20м×20м. В вершинах A и B размещены датчики, которые каждую секунду выдают квадрат расстояния до движущегося объекта в точке M_t. Известно, что объект движется по окружности с постоянной угловой скоростью. Обозначим $p_t = AM_t^2$ и $q_t = BM_t^2$. Время t измеряется в секундах. Результаты трех измерений приведены в таблице.

t	0	1	2
p_t	320	346	317,6
q_t	80	146	221,6

а) **(10 баллов)** Введите систему координат с началом в точке A, осью абсцисс по направлению луча AB и осью ординат по направлению луча AD (единица измерения в метрах). Найдите координаты точек M_0, M_1, M_2 в этой системе координат.

б) **(20 баллов)** Найдите уравнение окружности, по которой движется точка M_t.

в) **(20 баллов)** Какие значения p_3 и q_3 выдадут датчики в момент времени $t = 3$с ?



Решение

а) Для точки $M_t(x_t; y_t)$ выполняются равенства (1) $\begin{cases} x_t^2 + y_t^2 = p_t; \\ (20 - x_t)^2 + y_t^2 = q_t. \end{cases}$ Откуда получаем (2) $\begin{cases} x_t = 10 - \dfrac{p_t - q_t}{40}; \\ y_t = \sqrt{p_t - x_t^2}. \end{cases}$ Подставляя значения из таблицы приходим к ответу.

б) Общий вид уравнения окружности с центром в точке $O(a; b)$ и радиусом r: $(x - a)^2 + (y - b)^2 = r^2$. Вместо x и y подставляя координаты точек M_0, M_1, M_2 получаем систему уравнений с тремя неизвестными, решением которой является тройка $a = 11$, $b = 8$, $r = 5$.

в) Пусть угловая скорость объекта равна α. В момент времени t угол M_0OM_t равняется αt, $\cos\alpha = \cos\angle M_0OM_1 = \dfrac{4}{5}$ по теореме косинусов. Тогда вектор $\overrightarrow{OM_3}$ имеет координаты $\{5\cos 3\alpha; 5\sin 3\alpha\}$; $\cos 3\alpha = 4\cos^3\alpha - 3\cos\alpha = -0,352$, $\sin 3\alpha = 3\sin\alpha - 4\sin^3\alpha = 0,936$. $M_3 = (11 + 5\cdot(-0,352); 8 + 5\cdot 0,936) = (9,24; 12,68)$ подставляя в формулы (1) получаем ответ.

Дополнительные критерии оценки

Отсутствуют.

3.3. Задачи по информатике

Задача 3.3.1. Траектории (15 баллов)

Уборка - дело важное, вот только никому не нравится ей заниматься. И Сереже тоже! Но ему повезло - недавно его мама подарила ему набор из большого количества роботов, которые могут делать за Сережу работу по дому.

Однако, радость Сережи была не долгой. Он совершенно не представляет, как ими пользоваться. Меню настроек - сложное, а инструкция - на английском.

Серёжа знает, что вы увлекаетесь робототехникой, и обратился к вам за помощью.

Через пару часов началось тестирование: роботы носились по всей квартире. Бесцельно, но весело.

Оказалось, что каждый робот двигается по заранее определенной (запрограммированной по умолчанию) траектории. Траектория движения каждого из роботов представляет собой ломаную. А сами роботы имеют цилиндрическую форму и убирают весь мусор, который окажется под ними в какой-либо момент времени.

Для начала Сережа поручил вам определить пересекаются ли траектории двух роботов.

Формат входных данных

В первой строке два целых числа:

- $2 \leqslant n \leqslant 500$ - количество точек в траектории первого робота;
- $2 \leqslant m \leqslant 500$ - количество точек в траектории второго робота.

В следующих двух строках заданы точки ($2 \cdot n$ целых чисел в первой строке и $2 \cdot m$ целых чисел во второй строке): $x_{i,j}$ $y_{i,j}$ — j-я точка траектории i-го робота ($|x_{i,j}| \leqslant 10^9$; $|y_{i,j}| \leqslant 10^9$).

Никакие две последовательные точки никакой из траекторий не совпадают.

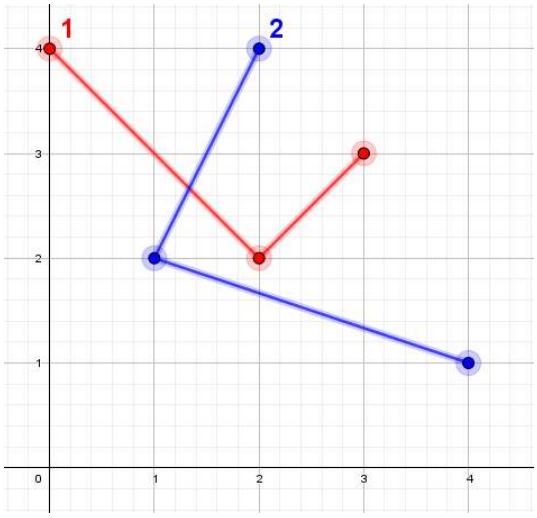


Рис. 1.1: Рисунок к первому примеру

Формат выходных данных

Выведите *Yes* , если траектории пересекаются, иначе - *No*.

Примеры

Пример №1

Стандартный ввод
3 3 0 4 2 2 3 3 2 4 1 2 4 1
Стандартный вывод
Yes

Пример №2

Стандартный ввод
2 2 0 0 1 1000000000 1 999999999 2 999999999
Стандартный вывод
No

Способ оценки работы

За решение задачи начислялось:

- **5 баллов**, если пройдена только первая группа тестов;
- **10 баллов**, если пройдена первая и вторая группы тестов;
- **15 баллов**, если пройдены все тесты.

Для проверки результата использовался следующий код на языке Python3

```
def generate():
    return []

def check(reply, clue):
    return reply.strip() == clue.strip()

x = []
y = []

# triangle area
def area(a,b,c):
    s = (x[b] - x[a]) * (y[c] - y[a]) - (y[b] - y[a]) * (x[c] - x[a])
    return -1 if s < 0 else (1 if s > 0 else 0)

# bounding box
def box(a, b, c, d):
    return max(min(a, b), min(c, d)) <= min(max(a, b), max(c, d))
```

```
# пересечение отрезков
def intersect(a, b, c, d):
    return box(x[a],x[b],x[c],x[d]) &
           box(y[a],y[b],y[c],y[d]) &
           (area(a, b, c) * area(a, b, d) <= 0) &
           (area(c, d, a) * area(c, d, b) <= 0)

def solve(dataset):
    ds = dataset.splitlines()

    s = ds[0].split()
    n = int(s[0])
    m = int(s[1])

    s = ds[1].split()
    for i in range(n):
        x.append(int(s[i*2]))
        y.append(int(s[i*2+1]))

    s = ds[2].split()
    for i in range(m):
        x.append(int(s[i * 2]))
        y.append(int(s[i * 2 + 1]))

    inter = False

    for i in range(n - 1):
        for j in range(m - 1):
            inter |= intersect(i, i + 1, n + j, n + j + 1)

    return "Yes" if inter else "No"
```

Решение

В данной задаче достаточно проверить пересекается ли хотя бы один отрезок первой траектории с хотя бы одним отрезком второй траектории.

Асимптотика: $O(n \cdot m)$

Пример программы

Ниже представлено решение на языке Python3

```
x = []
y = []

# triangle area
def area(a,b,c):
    s = (x[b] - x[a]) * (y[c] - y[a]) - (y[b] - y[a]) * (x[c] - x[a])
    return -1 if s < 0 else (1 if s > 0 else 0)

# bounding box
def box(a, b, c, d):
    return max(min(a, b), min(c, d)) <= min(max(a, b), max(c, d))


# пересечение отрезков
```

```
def intersect(a, b, c, d):
    return box(x[a],x[b],x[c],x[d]) & box(y[a],y[b],y[c],y[d]) \
    & (area(a, b, c) * area(a, b, d) <= 0) \
    & (area(c, d, a) * area(c, d, b) <= 0)

def solve(dataset):
    ds = dataset.splitlines()

    s = ds[0].split()
    n = int(s[0])
    m = int(s[1])

    s = ds[1].split()
    for i in range(n):
        x.append(int(s[i * 2]))
        y.append(int(s[i * 2 + 1]))

    s = ds[2].split()
    for i in range(m):
        x.append(int(s[i * 2]))
        y.append(int(s[i * 2 + 1]))

    inter = False

    for i in range(n - 1):
        for j in range(m - 1):
            inter |= intersect(i, i + 1, n + j, n + j + 1)

    return "Yes" if inter else "No"

print(solve(input() + '\n' + input() + '\n' + input()))
```

Задача 3.3.2. Радиус робота-уборщика (20 баллов)

Пока вы занимались проверкой пересечения траекторий, Серёжа не терял времени даром. Он разработал идеальную (по его мнению) траекторию движения робота уборщика. Учёл (по его мнению) все факторы: расположение мебели, обуви и т. д.

И теперь он просит вас рассчитать, какой должен быть минимальный радиус робота, чтобы он смог убрать весь мусор. При этом робот должен двигаться по траектории, разработанной Серёжей, а мусор - это набор из m материальных точек.

Формат входных данных

В первой строке два целых числа:

- $2 \leqslant n \leqslant 500$ - количество точек в траектории робота;
- $1 \leqslant m \leqslant 500$ - количество мусор-точек.

В следующей строке заданы точки ($2 \cdot n$ целых чисел): x_i y_i — i-я точка траектории робота ($1 \leqslant i \leqslant n$; $|x_i| \leqslant 10^9$; $|y_i| \leqslant 10^9$).

В следующей строке заданы точки ($2 \cdot m$ целых чисел): x_j y_j — j-я точка мусора ($1 \leqslant j \leqslant m$; $|x_j| \leqslant 10^9$; $|y_j| \leqslant 10^9$).

Никакие две последовательные точки траекторий не совпадают.

Формат выходных данных

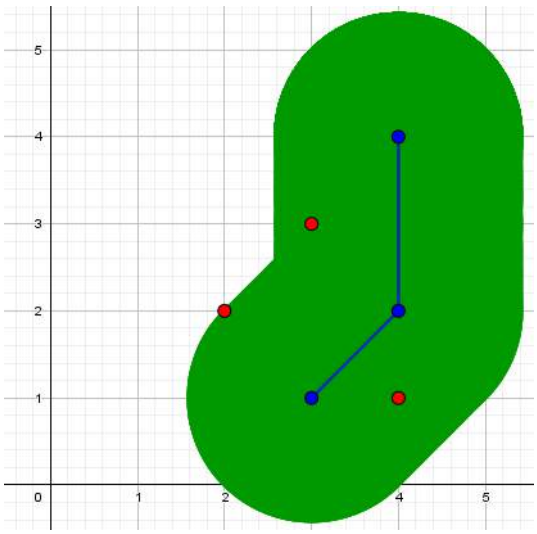


Рис. 1.2: Рисунок к первому примеру

Одно неотрицательное число – минимальный радиус.

Ваш ответ будет засчитан, если его абсолютная или относительная ошибка не превосходит 10^{-6}. Формально, пусть ваш ответ равен a, а ответ жюри равен b. Ваш ответ будет засчитан, если $\frac{|a-b|}{\max(1,b)} \leqslant 10^{-6}$.

Примеры

Пример №1

Стандартный ввод
3 3 4 4 4 2 3 1 4 1 2 2 3 3
Стандартный вывод
1.4142135623730951

Пример №2

Стандартный ввод
2 1 0 0 1 1 1 1
Стандартный вывод
0.0

Способ оценки работы

За решение задачи начислялось:

- **5 баллов**, если пройдена только первая группа тестов;
- **5 баллов**, если пройдена первая и вторая группы тестов;

- **10 баллов**, если пройдены все тесты.

Для проверки результата использовался следующий код на языке Python3

```
def generate():
    return []

def check(reply, clue):
    a = float(reply)
    b = float(clue)
    return abs(a - b) / max(1.0, b) <= 1e-6

from collections import namedtuple
from math import sqrt

Point = namedtuple("Point", ["x", "y"])

# расстояние между точками a и b
def dist(a, b):
        return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y))

# расстояние между точкой c и отрезком ab
def dist2(a, b, c):
        A = a.y - b.y
        B = b.x - a.x
        C = b.y * a.x - a.y * b.x
        cc = B * c.x - A * c.y
        d = A * A + B * B
        p = Point((cc * B - C * A) / d, -(cc * A + C * B) / d)

        return dist(c, p) if dist(a, p) + dist(p, b) - dist(a, b) < 1e-7 else min(dist(c, a), dist(c

def solve(dataset):
    ds = dataset.splitlines()

    s = ds[0].split()
    n = int(s[0])
    m = int(s[1])

    s = ds[1].split()
    p = []
    for i in range(n):
        p.append(Point(int(s[i * 2]), int(s[i * 2 + 1])))

    s = ds[2].split()
    pm = []
    for i in range(m):
        pm.append(Point(int(s[i * 2]), int(s[i * 2 + 1])))

    dist = [1e10] * m

    for i in range(n - 1):
        for j in range(m):
            dist[j] = min(dist[j], dist2(p[i], p[i + 1], pm[j]))

    ans = dist[m-1]
    for i in range(m - 1):
        ans = max(ans, dist[i])

    return str(ans)
```

Решение

В данной задаче достаточно найти для каждой мусор-точки расстояние до ближайшего к ней отрезка, после чего найти максимум среди этих значений – это и будет ответ.

Асимптотика: $O(n \cdot m)$

Пример программы

Ниже представлено решение на языке Python3

```
from collections import namedtuple
from math import sqrt

Point = namedtuple("Point", ["x", "y"])


# расстояние между точками a и b
def dist(a, b):
        return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y))


# расстояние между точкой c и отрезком ab
def dist2(a, b, c):
        A = a.y - b.y
        B = b.x - a.x
        C = b.y * a.x - a.y * b.x
        cc = B * c.x - A * c.y
        d = A * A + B * B
        p = Point((cc * B - C * A) / d, -(cc * A + C * B) / d)

        return dist(c, p) if dist(a, p) + dist(p, b) - dist(a, b) < 1e-7
                          else min(dist(c, a), dist(c, b))


def solve(dataset):
    ds = dataset.splitlines()

    s = ds[0].split()
    n = int(s[0])
    m = int(s[1])

    s = ds[1].split()
    p = []
    for i in range(n):
        p.append(Point(int(s[i * 2]), int(s[i * 2 + 1])))

    s = ds[2].split()
    pm = []
    for i in range(m):
        pm.append(Point(int(s[i * 2]), int(s[i * 2 + 1])))

    dist = [1e10] * m

    for i in range(n - 1):
        for j in range(m):
            dist[j] = min(dist[j], dist2(p[i], p[i + 1], pm[j]))
```

```
47
48      ans = dist[m-1]
49      for i in range(m - 1):
50          ans = max(ans, dist[i])
51
52      return ans
53
54
55  print(solve(input() + '\n' + input() + '\n' + input()))
```

Задача 3.3.3. Радиус роботов-уборщиков (25 баллов)

Не успели вы закончить с предыдущей задачей, как Серёжа уже нарисовал какие-то ломаные на полу и сказал, что роботы должны двигаться по ним, не сталкиваться между собой и при этом должны быть одинаковых размеров (по каждой из траекторий движется только один робот; касание не считается столкновением).

Рассчитайте максимальный радиус, при котором роботы не столкнутся ни при какой разнице времён запуска роботов.

Формат входных данных

В первой строке два целых числа:

- $2 \leqslant n \leqslant 500$ - количество роботов;
- $2 \leqslant m \leqslant 500$ - количество точек в траектории каждого робота.

При этом $4 \leqslant n \cdot m \leqslant 500$.

В следующих n строках задано по m точек ($2 \cdot m$ целых числа):$x_{i,j}$, $y_{i,j}$ — j-я точка траектории i-го робота ($1 \leqslant i \leqslant n$; $1 \leqslant j \leqslant m$; $|x_{i,j}| \leqslant 10^9$; $|y_{i,j}| \leqslant 10^9$).

Никакие две последовательные точки никакой из траекторий не совпадают.

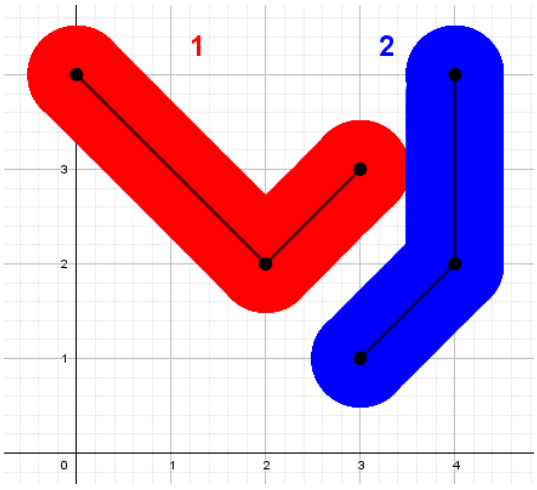


Рис. 1.3: Рисунок к первому примеру

Формат выходных данных

Если есть пересекающиеся траектории, выведите -1, иначе одно неотрицательное число – максимальный радиус.

Ваш ответ будет засчитан, если его абсолютная или относительная ошибка не превосходит 10^{-6}. Формально, пусть ваш ответ равен a, а ответ жюри равен b. Ваш ответ будет засчитан, если $\frac{|a-b|}{\max(1,b)} \leqslant 10^{-6}$.

Примеры

Пример №1

Стандартный ввод
2 3 0 4 2 2 3 3 4 4 4 2 3 1
Стандартный вывод
0.5

Способ оценки работы

За решение задачи начислялось:

- **10 баллов**, если пройдена только первая группа тестов;
- **25 баллов**, если пройдены все тесты.

Для проверки результата использовался следующий код на языке Python3

```
def generate():
    return []

def check(reply, clue):
    a = float(reply)
    b = float(clue)
    return abs(a - b) / max(1.0, b) <= 1e-6

from collections import namedtuple
from math import sqrt

Point = namedtuple("Point", ["x", "y"])

# triangle area
def area(a, b, c):
    s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x)
    return -1 if s < 0 else (1 if s > 0 else 0)

# bounding box
def box(a, b, c, d):
    return max(min(a, b), min(c, d)) <= min(max(a, b), max(c, d))

# пересечение отрезков
def intersect(a, b, c, d):
    return box(a.x, b.x, c.x, d.x) & box(a.y, b.y, c.y, d.y) \
           & (area(a, b, c) * area(a, b, d) <= 0) & (area(c, d, a) * area(c, d, b) <= 0)

# расстояние между точками a и b
def dist(a, b):
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y))
```

```
# расстояние между точкой c и отрезком ab
def dist2(a, b, c):
    A = a.y - b.y
    B = b.x - a.x
    C = b.y * a.x - a.y * b.x
    cc = B * c.x - A * c.y
    d = A * A + B * B
    p = Point((cc * B - C * A) / d, -(cc * A + C * B) / d)

    return dist(c, p) if dist(a, p) + dist(p, b) - dist(a, b) < 1e-7 else min(dist(c, a), dist(c, b))

# расстояние между отрезками ab и cd
def dist3(a, b, c, d):
    return min(min(dist2(c, d, a), dist2(c, d, b)), min(dist2(a, b, c), dist2(a, b, d)))

def solve(dataset):
    ds = dataset.splitlines()

    s = ds[0].split()
    n = int(s[0])
    m = int(s[1])

    p = [[]]
    for i in range(n):
        s = ds[i + 1].split()
        p.append([])
        for j in range(m):
            p[i].append(Point(int(s[j * 2]), int(s[j * 2 + 1])))

    inter = False

    for i in range(n):
        for j in range(i + 1, n):
            for ii in range(m - 1):
                for jj in range(m - 1):
                    inter |= intersect(p[i][ii], p[i][ii + 1], p[j][jj], p[j][jj + 1])

    if inter:
        return str(-1)

    ans = 1e15

    for i in range(n):
        for j in range(i + 1, n):
            for ii in range(m - 1):
                for jj in range(m - 1):
                    ans = min(ans, dist3(p[i][ii], p[i][ii + 1], p[j][jj], p[j][jj + 1]))

    return str(ans / 2.0)
```

Решение

В данной задаче достаточно проверить есть ли пересекающиеся траектории (аналогично тому, как это делается в задаче A) и, если такие найдутся, вывести -1 или, если таких траекторий нет, найти два отрезка, которые принадлежат разным траекториям и расстояние между которыми минимально, и вывести поделенное на два расстояние между этими отрезками (тут пригодится решение задачи B).

Асимптотика: $O((n \cdot m)^2)$

Пример программы

Ниже представлено решение на языке Python3

```
from collections import namedtuple
from math import sqrt

Point = namedtuple("Point", ["x", "y"])


# triangle area
def area(a, b, c):
    s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x)
    return -1 if s < 0 else (1 if s > 0 else 0)


# bounding box
def box(a, b, c, d):
    return max(min(a, b), min(c, d)) <= min(max(a, b), max(c, d))


# пересечение отрезков
def intersect(a, b, c, d):
    return box(a.x, b.x, c.x, d.x) & box(a.y, b.y, c.y, d.y) \
           & (area(a, b, c) * area(a, b, d) <= 0) & (area(c, d, a) * area(c, d, b) <= 0)


# расстояние между точками a и b
def dist(a, b):
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y))


# расстояние между точкой c и отрезком ab
def dist2(a, b, c):
    A = a.y - b.y
    B = b.x - a.x
    C = b.y * a.x - a.y * b.x
    cc = B * c.x - A * c.y
    d = A * A + B * B
    p = Point((cc * B - C * A) / d, -(cc * A + C * B) / d)

    return dist(c, p) if dist(a, p) + dist(p, b) - dist(a, b) < 1e-7
                      else min(dist(c, a), dist(c, b))


# расстояние между отрезками ab и cd
def dist3(a, b, c, d):
    return min(min(dist2(c, d, a), dist2(c, d, b)), min(dist2(a, b, c), dist2(a, b, d)))


def solve(dataset):
    ds = dataset.splitlines()

    s = ds[0].split()
    n = int(s[0])
    m = int(s[1])

    p = [[]]
    for i in range(n):
        s = ds[i + 1].split()
```

```
57          p.append([])
58          for j in range(m):
59              p[i].append(Point(int(s[j * 2]), int(s[j * 2 + 1])))
60
61      inter = False
62
63      for i in range(n):
64          for j in range(i + 1, n):
65              for ii in range(m - 1):
66                  for jj in range(m - 1):
67                      inter |= intersect(p[i][ii], p[i][ii + 1], p[j][jj], p[j][jj + 1])
68
69      if inter:
70          return -1
71
72      ans = 1e15
73
74      for i in range(n):
75          for j in range(i + 1, n):
76              for ii in range(m - 1):
77                  for jj in range(m - 1):
78                      ans = min(ans, dist3(p[i][ii], p[i][ii + 1], p[j][jj], p[j][jj + 1]))
79
80      return ans / 2.0
81
82
83  pds = input()
84  n = int(pds.split()[0])
85  for i in range(n):
86      pds += '\n'
87      pds += input()
88
89  print(solve(pds))
```

Задача 3.3.4. Уборка (15 баллов)

Определить возможность столкновения - задача довольно сложная, поэтому для начала вы договорились с Серёжей, что перепрограммируете роботов (зададите им новые траектории) и будете запускать их по очереди.

Происходит это следующим образом: очередного робота Серёжа ставит на первую точку траектории и запускает. После достижения конечной точки Серёжа моментально убирает робота и переходит к следующему.

Ваша задача определить каких роботов нужно запустить, чтобы убрать весь мусор. При этом нужно минимизировать количество запущенных роботов.

Допускается погрешность не более 10^{-6} при расчёте расстояния от центра робота в любой момент времени до любой из мусор-точек.

Формат входных данных

В первой строке три целых числа:

- $1 \leqslant n \leqslant 20$ - количество роботов;
- $2 \leqslant m \leqslant 500$ - количество точек в траектории каждого робота;
- $1 \leqslant k \leqslant 200$ - количество мусор-точек.

В следующих n строках по $2 \cdot m + 1$ целых чисел: $1 \leqslant radius_i \leqslant 10^9$ - радиус i-го робота и m точек ($2 \cdot m$ целых числа):$x_{i,j}$, $y_{i,j}$ — j-я точка траектории i-го робота ($1 \leqslant i \leqslant n$; $1 \leqslant j \leqslant m$; $|x_{i,j}| \leqslant 10^9$; $|y_{i,j}| \leqslant 10^9$).

В последней строке заданы мусор-точки ($2 \cdot k$ целых чисел): x_i y_i — i-я точка мусора ($1 \leqslant i \leqslant k$; $|x_i| \leqslant 10^9$; $|y_i| \leqslant 10^9$).

Никакие две последовательные точки никакой из траекторий не совпадают.

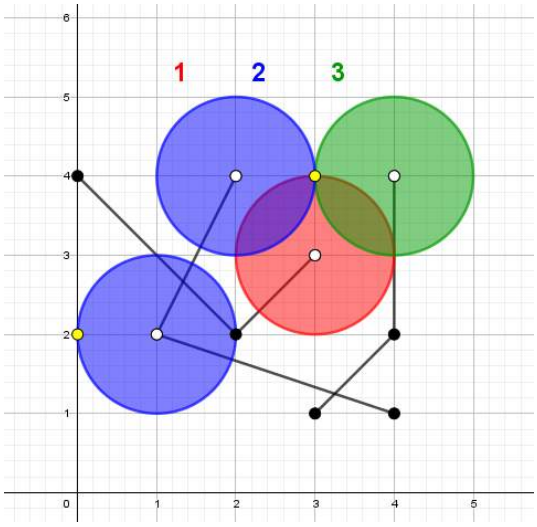


Рис. 1.4: Рисунок к первому примеру

Формат выходных данных

Если весь мусор невозможно убрать, выведите -1, иначе одно положительное целое число — минимальное необходимое для уборки всего мусора количество роботов.

Примеры

Пример №1

Стандартный ввод
3 3 2 1 0 4 2 2 3 3 1 2 4 1 2 4 1 1 4 4 4 2 3 1 0 2 3 4

Стандартный вывод
1

Способ оценки работы

За решение задачи начислялось:

- **15 баллов**, если пройдены все тесты.

Для проверки результата использовался следующий код на языке Python3

```python
def generate():
    return []

def check(reply, clue):
    return reply.strip() == clue.strip()
```

Решение

С помощью функции нахождения расстояния от точки до отрезка можно определить для каждого робота какие мусор-точки он может собрать.

Далее достаточно сделать полный перебор (битмасок) вариантов очереди запуска роботов и найти тот вариант, в котором используется минимум роботов и весь мусор убран.

Асимптотика: $O(n \cdot k \cdot (2^n + m))$

Пример программы

Ниже представлено решение на языке Java

```java
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.StringTokenizer;

import static java.lang.Math.min;
import static java.lang.Math.sqrt;

public class Main {
    // Уборка
    private FastScanner in;
    private PrintWriter out;

    class Point {
        double x, y;

        Point(double x, double y) {
            this.x = x;
            this.y = y;
        }
    }

    private double eps = 1e-7;

    // расстояние между точками a и b
    private double dist(Point a, Point b) {
        return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
    }

    // расстояние между точкой c и отрезком ab
    private double dist(Point a, Point b, Point c) {
        double A = a.y - b.y, B = b.x - a.x, C = b.y * a.x - a.y * b.x;
        double cc = B * c.x - A * c.y, d = A * A + B * B;
        Point p = new Point((cc * B - C * A) / d, -(cc * A + C * B) / d);
```

```
37
38          return dist(a, p) + dist(p, b) - dist(a, b) < eps ?
39                  dist(c, p) :
40                  min(dist(c, a), dist(c, b));
41      }
42
43      // траектории и радиусы роботов
44      private int n, m; // количество роботов и количество точек в траектории каждого робота
45      private double[] radius; // радиусы роботов
46      private Point[][] p; // все точки всех траекторий
47
48      // мусор
49      private int k;
50      private Point[] pk;
51
52      // инициализация
53      private void init() throws IOException {
54          n = in.nextInt();
55          m = in.nextInt();
56          k = in.nextInt();
57
58          radius = new double[n];
59          p = new Point[n][m];
60          pk = new Point[k];
61
62          for (int i = 0; i < n; i++) {
63              radius[i] = in.nextInt();
64
65              for (int j = 0; j < m; j++)
66                  p[i][j] = new Point(in.nextInt(), in.nextInt());
67          }
68
69          for (int i = 0; i < k; i++)
70              pk[i] = new Point(in.nextInt(), in.nextInt());
71      }
72
73      // Основа решения
74      private void solve() throws IOException {
75          boolean[][] can = new boolean[n][k];
76
77          for (int i = 0; i < n; i++)
78              for (int ki = 0; ki < k; ki++)
79                  for (int j = 0; j + 1 < m && !can[i][ki]; j++)
80                      can[i][ki] = dist(p[i][j], p[i][j + 1], pk[ki]) - radius[i] < eps;
81
82          int full = 1 << n, ans = n + 1;
83          boolean ok;
84          for (int f = 1; f < full; f++) {
85              ok = true;
86              for (int ki = 0; ki < k && ok; ki++) {
87                  ok = false;
88                  for (int i = 0; i < n && !ok; i++)
89                      ok = ((1 << i) & f) > 0 && can[i][ki];
90              }
91              if (ok)
92                  ans = min(ans, cnt(f));
93          }
94
95          out.println(ans > n ? -1 : ans);
96      }
```

```
97
98      private int cnt(int f) {
99          int cnt = 0;
100         for (char c : Integer.toBinaryString(f).toCharArray())
101             cnt += c == '1' ? 1 : 0;
102         return cnt;
103     }
104
105     class FastScanner {
106         StringTokenizer st;
107         BufferedReader br;
108
109         FastScanner(InputStream s) {
110             br = new BufferedReader(new InputStreamReader(s));
111         }
112
113         String next() throws IOException {
114             while (st == null || !st.hasMoreTokens())
115                 st = new StringTokenizer(br.readLine());
116             return st.nextToken();
117         }
118
119         int nextInt() throws IOException {
120             return Integer.parseInt(next());
121         }
122     }
123
124     private void run() throws IOException {
125         in = new FastScanner(System.in);
126         out = new PrintWriter(System.out);
127
128         init();
129         solve();
130
131         out.flush();
132         out.close();
133     }
134
135     public static void main(String[] args) throws IOException {
136         new Main().run();
137     }
138 }
```

Задача 3.3.5. Столкновения (15 баллов)

Поскольку Серёжа не может перепрограммировать роботов, Серёжа решил переставить их так, чтобы они собрали больше мусора, тем самым всё же изменив траектории их движения. Сейчас он готовится запустить всех роботов и посмотреть, сколько мусора они уберут. Но вы-то знаете, что теперь эти роботы не только могут убрать мусор, но и врезаться друг в друга!

Вам нужно срочно указать Серёже какие пары роботов могут столкнуться, если их запустить одновременно.

После достижения конечной точки Серёжа моментально убирает робота.

Касание (расстояние меньше 10^{-6}) считайте столкновением.

Формат входных данных

В первой строке два целых числа:

- $2 \leqslant n \leqslant 100$ - количество роботов;
- $2 \leqslant m \leqslant 100$ - количество точек в траектории каждого робота.

В следующих n строках по $2 \cdot m + 2$ целых чисел: $1 \leqslant radius_i \leqslant 10^9$ - радиус i-го робота, $1 \leqslant speed_i \leqslant 10^9$ - скорость i-го робота и m точек $x_{i,j}$, $y_{i,j}$ — j-я точка траектории i-го робота ($1 \leqslant i \leqslant n$; $1 \leqslant j \leqslant m$; $|x_{i,j}| \leqslant 10^9$; $|y_{i,j}| \leqslant 10^9$).

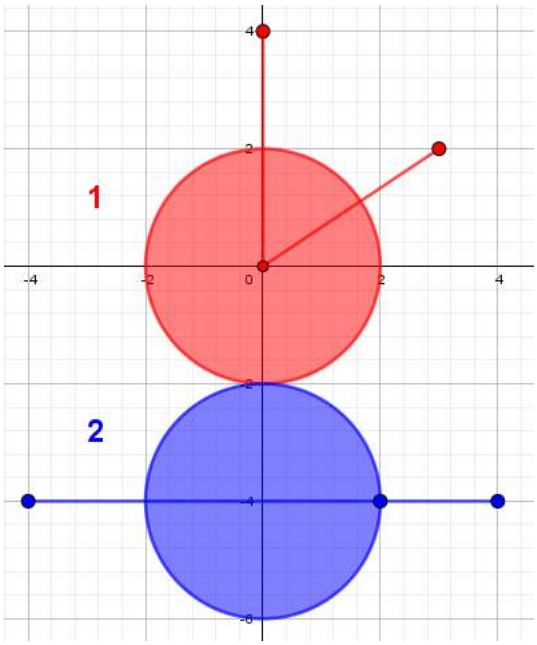


Рис. 1.5: Рисунок к первому примеру. Столкновение происходит примерно через 4 секунды после запуска.

Формат выходных данных

В первой строке одно целое число: k – число пар роботов, которые столкнутся, если запустить одновременно.

В следующих k строках по одной паре целых чисел: i j - i-й робот столкнётся с j-м ($1 \leqslant i \leqslant n$; $1 \leqslant j \leqslant n$; $i < j$)

Примеры

Пример №1

Стандартный ввод
2 3 2 1 0 4 0 0 3 2 2 1 -4 -4 2 -4 4 -4

Стандартный вывод
1 1 2

Способ оценки работы

За решение задачи начислялось:

- **15 баллов**, если пройдены все тесты.

Для проверки результата использовался следующий код на языке Python3

```
def generate():
    return []

def check(reply, clue):
    return reply.strip() == clue.strip()
```

Решение

Для решения данной задачи нужно написать проверку на столкновение двух одновременно запущенных роботов.

Можно заметить, что в промежутки времени между достижениями роботами точек роботы движутся равномерно и прямолинейно. А это значит, что с помощью тернарного поиска на каждом из таких промежутков времени, мы можем найти минимальное расстояние между центрами роботов, достижимое в некоторый текущий промежуток времени. Если хотя бы одно из этих минимальных расстояний не превышает суммы радиусов роботов, то столкновения не избежать.

Асимптотика: $O(n^2 \cdot m)$

Пример программы

Ниже представлено решение на языке Java

```
1  import java.io.BufferedReader;
2  import java.io.InputStream;
3  import java.io.InputStreamReader;
4  import java.io.IOException;
5  import java.io.PrintWriter;
6  import java.util.StringTokenizer;
7
8  import static java.lang.Math.min;
9  import static java.lang.Math.sqrt;
10
11 public class Main {
12     // Столкновения
13     private FastScanner in;
14     private PrintWriter out;
15
16     class Point {
17         double x, y;
18
19         Point(double x, double y) {
20             this.x = x;
21             this.y = y;
22         }
23     }
24
```

```
25      private double eps = 1e-7;
26
27      // расстояние между точками a и b
28      private double dist(Point a, Point b) {
29          return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
30      }
31
32      // точка, в которой находился бы робот, если бы
33      // прошёл растояние s из точки a в направлении точки b
34      private Point goTo(Point a, Point b, double s) {
35          double S = dist(a, b);
36          return new Point(a.x + (b.x - a.x) / S * s, a.y + (b.y - a.y) / S * s);
37      }
38
39      // траектории и радиусы роботов
40      private int n, m; // количество роботов и количество точек в траектории каждого робота
41      private double[] radius, speed; // радиусы и скорости роботов
42      private Point[][] p; // все точки всех траекторий
43      private double[][] time;
44
45      // инициализация
46      private void init() throws IOException {
47          n = in.nextInt();
48          m = in.nextInt();
49
50          radius = new double[n];
51          speed = new double[n];
52          p = new Point[n][m];
53          time = new double[n][m];
54
55          for (int i = 0; i < n; i++) {
56              radius[i] = in.nextInt();
57              speed[i] = in.nextInt();
58
59              for (int j = 0; j < m; j++)
60                  p[i][j] = new Point(in.nextInt(), in.nextInt());
61          }
62      }
63
64      // заполняем таблицу времени
65      private void fillTime() {
66          for (int i = 0; i < n; i++) {
67              time[i][0] = 0.0;
68
69              for (int j = 0; j + 1 < m; j++)
70                  time[i][j + 1] = time[i][j] + dist(p[i][j], p[i][j + 1]) / speed[i];
71          }
72      }
73
74      // Основа решения
75      private void solve() throws IOException {
76          StringBuilder ans = new StringBuilder();
77          int cnt = 0;
78
79          for (int i = 0; i < n; i++)
80              for (int j = i + 1; j < n; j++)
81                  if (crash(i, j)) {
82                      ans.append(i + 1).append(' ').append(j + 1).append('\n');
83                      cnt++;
84                  }
```

```
85
86          out.print(cnt + "\n" + ans);
87      }
88
89      // проверяем на столкновение i-ого и j-ого роботов
90      private boolean crash(int i, int j) {
91          double pt = 0.0, nt;
92          double l, r, t;
93          double tl, tr, dl, dr;
94          Point pil, pir, pjl, pjr;
95
96          for (int ii = 1, jj = 1; ii < m && jj < m; pt = nt) {
97              nt = min(time[i][ii], time[j][jj]);
98
99              if (nt - pt > eps) {
100
101                 // тернарный поиск
102                 l = pt;
103                 r = nt;
104                 while (r - l > eps) {
105
106                     tl = l + (r - l) / 3;
107                     tr = tl + (r - l) / 3;
108
109                     pil = goTo(p[i][ii - 1], p[i][ii], (tl - time[i][ii - 1]) * speed[i]);
110                     pir = goTo(p[i][ii - 1], p[i][ii], (tr - time[i][ii - 1]) * speed[i]);
111
112                     pjl = goTo(p[j][jj - 1], p[j][jj], (tl - time[j][jj - 1]) * speed[j]);
113                     pjr = goTo(p[j][jj - 1], p[j][jj], (tr - time[j][jj - 1]) * speed[j]);
114
115                     dl = dist(pil, pjl);
116                     dr = dist(pir, pjr);
117
118                     if (dl > dr)
119                         l = tl;
120                     else
121                         r = tr;
122                 }
123
124                 t = (l + r) / 2.0;
125                 pil = goTo(p[i][ii - 1], p[i][ii], (t - time[i][ii - 1]) * speed[i]);
126                 pjl = goTo(p[j][jj - 1], p[j][jj], (t - time[j][jj - 1]) * speed[j]);
127
128                 if (dist(pil, pjl) - radius[i] - radius[j] < eps)
129                     return true;
130             }
131
132             if (time[i][ii] - nt < eps)
133                 ii++;
134             if (time[j][jj] - nt < eps)
135                 jj++;
136         }
137
138         return false;
139     }
140
141     class FastScanner {
142         StringTokenizer st;
143         BufferedReader br;
144
```

```
145         FastScanner(InputStream s) {
146             br = new BufferedReader(new InputStreamReader(s));
147         }
148 
149         String next() throws IOException {
150             while (st == null || !st.hasMoreTokens())
151                 st = new StringTokenizer(br.readLine());
152             return st.nextToken();
153         }
154 
155         int nextInt() throws IOException {
156             return Integer.parseInt(next());
157         }
158     }
159 
160     private void run() throws IOException {
161         in = new FastScanner(System.in);
162         out = new PrintWriter(System.out);
163 
164         init();
165         fillTime();
166         solve();
167 
168         out.flush();
169         out.close();
170     }
171 
172     public static void main(String[] args) throws IOException {
173         new Main().run();
174     }
175 }
```

Задача 3.3.6. Complete cleaning (10 баллов)

Возможно вам надоело решать вспомогательные задачи. Обрадуем: теперь вы готовы решить изначальную задачу.

У вас есть n роботов, они двигаются по траекториям состоящим из m точек каждая. Вам нужно убрать как можно больше мусора используя роботов, но вы очень сильно дорожите своими роботами. Поэтому вы не хотите, чтобы во время движения они столкнулись.

Таким образом, вам нужно вывести минимальное количество роботов, при котором можно убрать максимально возможное количество мусора. При этом выбранные роботы запускаются одновременно и не сталкиваются друг с другом.

После достижения конечной точки робот моментально исчезает.

Касание (расстояние меньше 10^{-6}) считайте столкновением.

Формат входных данных

В первой строке три целых числа:

- $1 \leqslant n \leqslant 20$ - количество роботов;
- $2 \leqslant m \leqslant 500$ - количество точек в траектории каждого робота;
- $1 \leqslant k \leqslant 200$ - количество мусор-точек.

В следующих n строках по $2 \cdot m + 2$ целых чисел: $1 \leqslant radius_i \leqslant 10^9$ - радиус i-го робота, $1 \leqslant speed_i \leqslant 10^9$ - скорость i-го робота и m точек $x_{i,j}$, $y_{i,j}$ — j-я точка траектории i-го робота ($1 \leqslant i \leqslant n$; $1 \leqslant j \leqslant m$; $|x_{i,j}| \leqslant 10^9$; $|y_{i,j}| \leqslant 10^9$).

В последней строке заданы мусор-точки ($2 \cdot k$ целых чисел): x_i y_i — i-я точка мусора ($1 \leqslant i \leqslant k$; $|x_i| \leqslant 10^9$; $|y_i| \leqslant 10^9$).

Формат выходных данных

Два неотрицательных числа - максимальное количество убранных мусор-точек и минимальное количество задействованных для этого роботов.

Примеры

Пример №1

Стандартный ввод
2 3 2 2 1 0 4 0 0 3 2 2 1 -4 -4 2 -4 4 -4 3 3 4 -6
Стандартный вывод
1 1

Пример №2

Стандартный ввод
2 3 2 2 1 0 4 0 0 3 2 2 1 -4 -4 2 -4 4 -4 0 -3 4 -6
Стандартный вывод
2 1

Способ оценки работы

За решение задачи начислялось:

- **10 баллов**, если пройдены все тесты.

Для проверки результата использовался следующий код на языке Python3

```
def generate():
    return []

def check(reply, clue):
    return reply.strip() == clue.strip()
```

Решение

Для решения последней задачи нужно оптимизировать фрагменты кода из прошлых задач.

Пример программы

Ниже представлено решение на языке Java

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.StringTokenizer;

import static java.lang.Math.min;
import static java.lang.Math.sqrt;

public class Main {
    // Complete cleaning
    private FastScanner in;
    private PrintWriter out;

    class Point {
        double x, y;

        Point(double x, double y) {
            this.x = x;
            this.y = y;
        }
    }

    private double eps = 1e-7;

    // расстояние между точками a и b
    private double dist(Point a, Point b) {
        return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
    }

    // расстояние между точкой c и отрезком ab
    private double dist(Point a, Point b, Point c) {
        double A = a.y - b.y, B = b.x - a.x, C = b.y * a.x - a.y * b.x;
        double cc = B * c.x - A * c.y, d = A * A + B * B;
        Point p = new Point((cc * B - C * A) / d, -(cc * A + C * B) / d);

        return dist(a, p) + dist(p, b) - dist(a, b) < eps ?
                   dist(c, p) :
                   min(dist(c, a), dist(c, b));
    }

    // точка, в которой находился бы робот, если бы
    // прошёл растояние s из точки a в направлении точки b
    private Point goTo(Point a, Point b, double s) {
        double S = dist(a, b);
        return new Point(a.x + (b.x - a.x) / S * s, a.y + (b.y - a.y) / S * s);
    }
```

```
50
51      // траектории и радиусы роботов
52      private int n, m; // количество роботов и количество точек в траектории каждого робота
53      private double[] radius, speed; // радиусы и скорости роботов
54      private Point[][] p; // все точки всех траекторий
55      private double[][] time;
56
57      // мусор
58      private int k;
59      private Point[] pk;
60
61      // инициализация
62      private void init() throws IOException {
63          n = in.nextInt();
64          m = in.nextInt();
65          k = in.nextInt();
66
67          radius = new double[n];
68          speed = new double[n];
69          p = new Point[n][m];
70          pk = new Point[k];
71
72          for (int i = 0; i < n; i++) {
73              radius[i] = in.nextInt();
74              speed[i] = in.nextInt();
75
76              for (int j = 0; j < m; j++)
77                  p[i][j] = new Point(in.nextInt(), in.nextInt());
78          }
79
80          for (int i = 0; i < k; i++)
81              pk[i] = new Point(in.nextInt(), in.nextInt());
82      }
83
84      // заполняем таблицу времени
85      private void fillTime() {
86          time = new double[n][m];
87
88          for (int i = 0; i < n; i++) {
89              time[i][0] = 0.0;
90
91              for (int j = 0; j + 1 < m; j++)
92                  time[i][j + 1] = time[i][j] + dist(p[i][j], p[i][j + 1]) / speed[i];
93          }
94      }
95
96      // Основа решения
97      private void solve() throws IOException {
98
99          // создаём матрицу столкновений
100         boolean[][] crash = new boolean[n][n];
101         for (int i = 0; i < n; i++)
102             for (int j = i + 1; j < n; j++)
103                 crash[i][j] = crash[j][i] = crash(i, j);
104
105         // создаём матрицу для проверки захвата роботами мусора
106         boolean[][] can = new boolean[n][k];
107         for (int i = 0; i < n; i++)
108             for (int ki = 0; ki < k; ki++)
109                 for (int j = 0; j + 1 < m && !can[i][ki]; j++)
```

```
                    can[i][ki] = dist(p[i][j], p[i][j + 1], pk[ki]) - radius[i] < eps;

        int full = 1 << n, ans = n;
        boolean ok;
        ArrayList<Integer> ids = new ArrayList<>();

        int max = 0, min = 0;
        // делаем полный перебор (битмасок) вариантов запуска роботов
        for (int f = 1; f < full; f++) {
            ids.clear();

            // создаём список роботов, которые запускаются
            for (int i = 0; i < n; i++)
                if (((1 << i) & f) > 0)
                    ids.add(i);

            // проверяем роботов из списка на столкновения
            ok = true;
            for (int i : ids)
                for (int j : ids)
                    ok &= !crash[i][j];

            // в случае столкновения, сразу переходим к следующему варианту
            if (!ok)
                continue;

            // считаем количество убираемых мусор-точек
            int cnt = 0;
            for (int ki = 0; ki < k; ki++) {
                ok = false;
                for (int i : ids)
                    if (can[i][ki]) {
                        ok = true;
                        break;
                    }

                if (ok)
                    cnt++;
            }

            // улучшаем текущий ответ
            if (cnt > max) {
                max = cnt;
                min = ids.size();
            } else if (cnt == max)
                if (ids.size() < min)
                    min = ids.size();
        }

        // выводим ответ: max - количество мусора, min - кол-во запущенных роботов
        out.println(max + " " + min);
    }

    // проверяем на столкновение i-ого и j-ого роботов
    private boolean crash(int i, int j) {
        double pt = 0.0, nt;
        double l, r, t;
        double tl, tr, dl, dr;
        Point pil, pir, pjl, pjr;

```

```
        for (int ii = 1, jj = 1; ii < m && jj < m; pt = nt) {
            nt = min(time[i][ii], time[j][jj]);

            if (nt - pt > eps) {

                // тернарный поиск
                l = pt;
                r = nt;
                while (r - l > eps) {

                    tl = l + (r - l) / 3;
                    tr = tl + (r - l) / 3;

                    pil = goTo(p[i][ii - 1], p[i][ii], (tl - time[i][ii - 1]) * speed[i]);
                    pir = goTo(p[i][ii - 1], p[i][ii], (tr - time[i][ii - 1]) * speed[i]);

                    pjl = goTo(p[j][jj - 1], p[j][jj], (tl - time[j][jj - 1]) * speed[j]);
                    pjr = goTo(p[j][jj - 1], p[j][jj], (tr - time[j][jj - 1]) * speed[j]);

                    dl = dist(pil, pjl);
                    dr = dist(pir, pjr);

                    if (dl > dr)
                        l = tl;
                    else
                        r = tr;
                }

                t = (l + r) / 2.0;
                pil = goTo(p[i][ii - 1], p[i][ii], (t - time[i][ii - 1]) * speed[i]);
                pjl = goTo(p[j][jj - 1], p[j][jj], (t - time[j][jj - 1]) * speed[j]);

                if (dist(pil, pjl) - radius[i] - radius[j] < eps)
                    return true;
            }
            if (time[i][ii] - nt < eps)
                ii++;
            if (time[j][jj] - nt < eps)
                jj++;
        }

        return false;
    }

    class FastScanner {
        StringTokenizer st;
        BufferedReader br;

        FastScanner(InputStream s) {
            br = new BufferedReader(new InputStreamReader(s));
        }

        String next() throws IOException {
            while (st == null || !st.hasMoreTokens())
                st = new StringTokenizer(br.readLine());
            return st.nextToken();
        }

        int nextInt() throws IOException {
            return Integer.parseInt(next());
```

```
230             }
231         }
232 
233         private void run() throws IOException {
234             in = new FastScanner(System.in);
235             out = new PrintWriter(System.out);
236 
237             init();
238             fillTime();
239             solve();
240 
241             out.flush();
242             out.close();
243         }
244 
245         public static void main(String[] args) throws IOException {
246             new Main().run();
247         }
248     }
```