§3 Заключительный этап: индивидуальная часть

3.1 Задачи по физике

Можно использовать решение одних подзадач для решения других. В этом случае отсутствие любых формул или рассуждений, которые уже были использованы для решения ранее, ошибкой не является.

Задача 1 (ориентация и навигация спутников)

Спутник движется по круговой орбите Земли, имеющей высоту h=700 км над уровнем моря. Масса Земли $5{,}97*10^{24}$ кг, а радиус 6371 км.

1) **(1 балл)** Найти угловую скорость движения спутника по орбите.

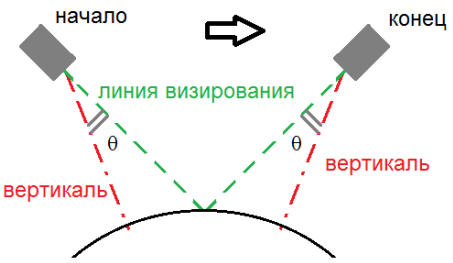
   Решение:
   Составим уравнение на основе закона всемирного тяготения:
   $$m \cdot \omega_{спутн}^2 \cdot (R_{зем} + h) = \frac{GmM_{зем}}{(R_{зем} + h)^2}$$
   Тогда для угловой скорости получаем
   $$\omega_{спутн} = \sqrt{\frac{G \cdot M_{зем}}{(R_{зем} + h)^3}} = \sqrt{\frac{6{,}67 \cdot 10^{-11} \cdot 5{,}97 \cdot 10^{24}}{(6371 + 700)^3 \cdot 10^9}} = 1{,}061 \cdot 10^{-3}\ \frac{радиан}{с}$$

2) **(2 балла)** Известно, что орбита этого спутника находится в экваториальной плоскости, а сам он движется с запада на восток. Найти скорость бега «тени спутника» на поверхности Земли (скорость подспутниковой точки) с учётом вращения Земли. Одни звёздные сутки составляют $T_{зем}$ = 23 часа 56 минут 4 секунды.

   Решение:
   Угловая скорость спутника:
   $$\omega_{спутн} = \frac{V_{орб}}{R_{зем} + h} = \sqrt{\frac{G \cdot M_{зем}}{(R_{зем} + h)^3}}$$
   Угловая скорость Земли:
   $$\omega_{зем} = \frac{2\pi}{T_{зем}}$$
   Тогда угловая скорость движения спутника относительно *вращающейся* Земли:
   $$\omega_{отн} = \omega_{спутн} - \omega_{зем}$$
   Из этого находим скорость тени спутника по поверхности Земли

$$V_{пст} = V_{тени} = \omega_{отн} \cdot R_{зем} = \omega_{спутн} \cdot R_{зем} - \frac{2\pi}{T_{зем}} \cdot R_{зем} = \omega_{спутн} \cdot R_{зем} - \frac{2\pi}{T_{зем}} \cdot R_{зем}$$
$$= 1{,}061 \cdot 10^{-3} \cdot 6371 \cdot 10^3 - \frac{6{,}28}{86164} \cdot 6371 \cdot 10^3 = 6761{,}396 - 464{,}346$$
$$= 6300 \ \frac{м}{с}$$

3) **(2 балла)** На спутнике размещена видеокамера, предназначенная для съёмки поверхности Земли. В штатном режиме камера снимает видеосюжет о площадке небольшого размера. При этом спутник «подворачивается», таким образом, изображение в кадре не двигается (подспутниковая точка движется, но конец линии визирования неподвижен). Известно, что ось камеры должна быть отклонена от вертикали не более чем на θ=5,7 градусов. Какова максимальная длительность видеосюжета?



Решение:
Угол θ мал, поэтому Землю будем считать плоской. За время съёмки подспутниковая точка пройдёт расстояние L и при этом
$$\frac{0{,}5 \cdot L}{h} = tg(\theta)$$
Тогда для времени съёмки T имеем:
$$T = \frac{L}{V_{пст}} = \frac{2 \cdot h \cdot tg(\theta)}{V_{пст}} = \frac{2 \cdot 7 \cdot 10^5 \cdot 0{,}1}{6300} = 22{,}2 \ с$$

4) **(5 баллов)** На спутнике размещена фотокамера, предназначенная для съёмки поверхности Земли. При этом оптическая ось объектива должна быть направлена вертикально вниз. Однако незадолго до подачи команды на съёмку в центре управления полётами обнаружили, что объектив камеры направлен **от** Земли, по радиусу орбиты. Для возвращения спутника в исходное состояние необходимо повернуть его вокруг продольной оси. Сколько времени займет переориентация спутника, если система управления может обеспечить угловое ускорение |ε|=0,0314 радиан/с^2 при перевороте спутника вокруг его продольной оси? Принять, что максимальная угловая скорость вращения спутника составляет ω_{max}=0,157 радиан/с. Изначально спутник не вращается вокруг продольной оси, во время съёмки этого тоже не должно быть.

Решение:
Сначала нужно проверить, будет ли превышена максимальная угловая скорость ω_{max} при **возможном** равноускоренном повороте до угла 90 градусов.
По аналогии с классическим равноускоренным движением имеем:
$$\omega\left(\frac{\pi}{2}\right) = \sqrt{2 \cdot |\varepsilon| \cdot \frac{\pi}{2}} = \sqrt{0{,}0314 \cdot 3{,}14} = 0{,}314 \ \frac{радиан}{с}$$
Так как ограничение по угловой скорости вращения спутника превышено, то циклограмма работы разделена на три временнЫх промежутка:
0…T1 – разгон с угловым ускорением ε от 0 до ω_{max},
T1…T2 – равномерное вращение с угловой скоростью ω_{max}

Т2…Т3 – торможение до нулевой угловой скорости с замедлением -ε

Найдем Т1:

$$T1 = T3 - T2 = \frac{\omega_{max}}{|\varepsilon|}$$

При этом поворот будет осуществлён на угол:

$$\Delta\varphi_{01} = \Delta\varphi_{23} = \frac{\omega_{max}^2}{2 \cdot |\varepsilon|} = 0{,}3925\ радиан = 22{,}48\ градусов$$

Находим угол, поворот на который будет проводиться при равномерном вращении Т1…Т2 и время этого вращения:

$$\Delta\varphi_{12} = \pi - 2 \cdot \Delta\varphi_{01} = \pi - \frac{\omega_{max}^2}{|\varepsilon|} = 2{,}355\ радиан$$

$$T2 - T1 = \frac{\Delta\varphi_{12}}{\omega_{max}} = \frac{\pi - \frac{\omega_{max}^2}{|\varepsilon|}}{\omega_{max}}$$

Тогда общее время переворота спутника Т3:

$$T_{переворот} = T3 = T1 + (T2 - T1) + (T3 - T2) = 2 \cdot T1 + (T2 - T1)$$
$$= 2 \cdot \frac{\omega_{max}}{|\varepsilon|} + \frac{\pi - \frac{\omega_{max}^2}{|\varepsilon|}}{\omega_{max}} = 2 \cdot \frac{0{,}157}{0{,}0314} + \frac{3{,}14 - \frac{0{,}157 \cdot 0{,}157}{0{,}0314}}{0{,}157}$$
$$= 2 \cdot 5 + \frac{2{,}355}{0{,}157} = 10 + 15 = 25\ с$$

5) **(10 баллов)** Орбита спутника такова, что он движется с запада на восток. Во время пролёта над городом Кито (столица Эквадора), расположенным почти точно на экваторе, возникла срочная необходимость провести съёмку остова Ямайка, находящегося в d=2000 км к северу от Кито (по прямой на поверхности Земли). Естественно, что осуществить это можно только при помощи наклона линии визирования (съёмка с креном). Определите требуемый угол крена и время переориентации спутника, если первоначально оптическая ось камеры была направлена вертикально вниз.

Решение:
При заданных условиях Землю уже нельзя считать плоской.
Найдём широту Ямайки θ:

$$\theta = \frac{d}{R_{зем}} = \frac{2000}{6371} = 0{,}314\ радиан = 17{,}98\ градусов$$

Тогда по теореме косинусов можно найти расстояние D от спутника до Ямайки:

$$D = \sqrt{(R_{зем} + h)^2 + R_{зем}^2 - 2 \cdot R_{зем} \cdot (R_{зем} + h) \cdot cos\theta}$$
$$= \sqrt{(6371 + 700)^2 + 6371^2 - 2 \cdot 6371 \cdot (6371 + 700) \cdot cos0{,}314}$$
$$= 2211{,}34\ км$$

Отсюда угол между вертикалью и оптической осью α определяем по теореме синусов (угол крена спутника):

$$\frac{D}{\sin\theta} = \frac{R_{зем}}{\sin\alpha}$$

$$\alpha = arcsin\left(\frac{R_{зем}}{D}\cdot \sin\theta\right) = arcsin\left(\frac{6371}{2211{,}34}\cdot \sin 0{,}314\right) = arcsin(0{,}8893)$$
$$= 62{,}79\ градусов$$

Находим время поворота спутника с положения «наблюдение вниз» до требуемого угла крена α. Угол α=62,79 градусов превышает более чем в два раза угол $\Delta\varphi_{01}$=22,48 градуса (из п.4). Следовательно, здесь также имеет место три временнЫх интервала работы системы ориентации спутника (а не два: разгон и замедление):
0…T1 – разгон с угловым ускорением ε от 0 до ω_{max},
T1…T2 – равномерное вращение с угловой скоростью ω_{max}
T2…T3 – торможение до нулевой угловой скорости с замедлением -ε

Аналогично п.4.

$$T_{переориент} = T3 = T1 + (T2 - T1) + (T3 - T2) = 2\cdot T1 + (T2 - T1)$$
$$= 2\cdot\frac{\omega_{max}}{|\varepsilon|} + \frac{\alpha - \frac{\omega_{max}^2}{|\varepsilon|}}{\omega_{max}} == 2\cdot\frac{0{,}157}{0{,}0314} + \frac{\frac{3{,}14}{180}\cdot 62{,}79 - \frac{0{,}157\cdot 0{,}157}{0{,}0314}}{0{,}157}$$
$$= 2\cdot 5 + \frac{0{,}31}{0{,}157} = 10 + 1{,}977 = 11{,}98\ c$$

Задача 2 (динамика спутников)

Спутник движется по круговой орбите Земли, имеющей высоту h=200 км над уровнем моря. Масса Земли $5{,}97*10^{24}$ кг, а радиус 6371 км.

1) **(1 балл)** Найти линейную скорость движения спутника по орбите.

Решение:
Составим уравнение на основе закона всемирного тяготения:
$$\frac{m\cdot V_{орб}^2}{R_{зем}+h} = \frac{GmM_{зем}}{(R_{зем}+h)^2}$$
Тогда для орбитальной скорости получаем
$$V_{орб} = \sqrt{\frac{G\cdot M_{зем}}{R_{зем}+h}} = \sqrt{\frac{6{,}67\cdot 10^{-11}\cdot 5{,}97\cdot 10^{24}}{(6371+200)\cdot 10^3}} = 7784\ \frac{м}{с}$$

2) **(2 балла)** Как известно, даже на высотах 200…300 км есть остаточная плотность атмосферы, вследствие чего возникает малое трение о воздух и спутник медленно теряет высоту орбиты. Какую работу должны совершить двигатели спутника, чтобы компенсировать потерю высоты орбиты? Нужно поднять высоту круговой орбиты с h_1=195 км до исходной h=200 км. Масса спутника m=1000 кг, изменением массы спутника вследствие расхода топлива пренебречь.

Решение:

Из закона сохранения механической энергии
$$E_2 = E_1 + A_{двиг}$$
С учётом этого получаем для работы двигателей:

$$A_{движ} = E_2 - E_1 = \left[-\frac{GmM_{зем}}{R_{зем}+h} + \frac{m \cdot V_2^2}{2}\right] - \left[-\frac{GmM_{зем}}{R_{зем}+h_1} + \frac{m \cdot V_1^2}{2}\right] =$$
$$= m \cdot \left[GM_{зем} \cdot \left(\frac{1}{R_{зем}+h_1} - \frac{1}{R_{зем}+h}\right) + \frac{1}{2} \cdot (V_2^2 - V_1^2)\right] =$$
$$= m \cdot \left[GM_{зем} \cdot \left(\frac{1}{R_{зем}+h_1} - \frac{1}{R_{зем}+h}\right) + \frac{1}{2}\left(\frac{GM_{зем}}{R_{зем}+h} - \frac{GM_{зем}}{R_{зем}+h_1}\right)\right] =$$
$$= m \cdot \left[GM_{зем} \cdot \frac{1}{2} \cdot \left(\frac{1}{R_{зем}+h_1} - \frac{1}{R_{зем}+h}\right)\right] =$$
$$= 10^3 \cdot \left[6{,}67 \cdot 10^{-11} \cdot 5{,}97 \cdot 10^{24} \cdot \frac{1}{2} \cdot 10^{-3}\left(\frac{1}{6371+195} - \frac{1}{6371+200}\right)\right]$$
$$= 23\ МДж$$

3) **(2 балла)** Оценить потерю высоты орбиты за 1 виток на высоте h=200 км вследствие вязкого трения об остаточную атмосферу. Принять, что на этой высоте плотность атмосферы составляет $2{,}52*10^{-10}$ кг/м^3. Площадь поперечного сечения спутника S=5 м^2. Масса спутника m=1000 кг.

Указание: сила аэродинамического сопротивления зависит от скорости V, площади поперечного сечения тела S и плотности газа ρ. Описывается формулой $F_{сопр} = -0{,}5*C_x*\rho*S*V^2$. *Для сильно разреженных газов коэффициент* $C_x=2$ *независимо от формы тела.*

Решение

Из закона сохранения механической энергии

$$E_2 = E_1 + A_{трения}$$

Тогда для полной энергии в начале и конце витка при потере высоты Δh имеем:

$$A_{трения} = E_2 - E_1$$

Получим:

$$A_{трения} = \left[-\frac{GmM_{зем}}{R_{зем}+h_2} + \frac{m \cdot V_2^2}{2}\right] - \left[-\frac{GmM_{зем}}{R_{зем}+h_1} + \frac{m \cdot V_1^2}{2}\right] =$$
$$= m \cdot \left[GM_{зем} \cdot \left(\frac{1}{R_{зем}+h_1} - \frac{1}{R_{зем}+h_2}\right) + \frac{1}{2}\left(\frac{GM_{зем}}{R_{зем}+h_2} - \frac{GM_{зем}}{R_{зем}+h_1}\right)\right]$$
$$= m \cdot \left[GM_{зем} \cdot \frac{1}{2} \cdot \left(\frac{1}{R_{зем}+h_1} - \frac{1}{R_{зем}+h_2}\right)\right]$$
$$= m \cdot \left[GM_{зем} \cdot \frac{1}{2} \cdot \left(\frac{1}{R_{зем}+h} - \frac{1}{R_{зем}+h-\Delta h}\right)\right]$$
$$= m \cdot \left[GM_{зем} \cdot \frac{1}{2} \cdot \left(\frac{-\Delta h}{(R_{зем}+h)(R_{зем}+h-\Delta h)}\right)\right] \simeq \frac{1}{2} \cdot mGM_{зем}\frac{-\Delta h}{(R_{зем}+h)^2}$$

Так как изменение орбиты незначительно, то пренебрежём изменением скорости спутника за один виток. Кроме того, при вычислении длины пути за один виток его можно считать окружностью. Тогда для силы сопротивления $F_{сопр}$ и пройдённого за виток пути L имеем:

$$A_{трения} = F_{сопр} \cdot L \simeq -\frac{1}{2} \cdot C_x \cdot \rho \cdot S \cdot V_{орб}^2 \cdot 2 \cdot \pi \cdot (R_{зем}+h)$$

Тогда составляем уравнением для потери высоты орбиты Δh:

$$\frac{1}{2}\cdot mGM_{зем}\frac{-\Delta h}{(R_{зем}+h)^2}=-\frac{1}{2}\cdot C_x\cdot\rho\cdot S\cdot V_{орб}^2\cdot 2\cdot\pi\cdot(R_{зем}+h)$$

Выражаем Δh:

$$\Delta h=\frac{C_x\cdot\rho\cdot S\cdot V_{орб}^2\cdot 2\cdot\pi\cdot(R_{зем}+h)^3}{mGM_{зем}}=\frac{1}{R_{зем}+h}\cdot\frac{C_x\cdot\rho\cdot S\cdot 2\cdot\pi\cdot(R_{зем}+h)^3}{m}$$
$$=\frac{C_x\cdot\rho\cdot 2\cdot S\cdot\pi\cdot(R_{зем}+h)^2}{m}$$
$$=\frac{2\cdot 2{,}52\cdot 10^{-10}\cdot 2\cdot 5\cdot 3{,}14\cdot(6371+200)^2\cdot 10^6}{10^3}=683\ м$$

4) **(5 баллов)** Оцените время существования спутника при начальной высоте орбиты 200 км. Трение об остаточную атмосферу описано в предыдущем пункте. Принять, что после снижения до критической высоты $h_{кр}$= 150 км торможение будет настолько сильным, что временем дальнейшего падения можно пренебречь по сравнению со временем снижения до высоты $h_{кр}$. Плотность воздуха на высоте 150 км составляет $2*10^{-9}$ кг/м^3, а в рассматриваемом диапазоне высот зависимость плотности воздуха от высоты принять линейной.

Решение:
Аналогично предыдущему пункту, оценим снижение орбиты за один виток на высоте 150 км.

$$\Delta h=\frac{C_x\cdot\rho\cdot 2\cdot S\cdot\pi\cdot(R_{зем}+h)^2}{m}=\frac{2\cdot 2\cdot 10^{-9}\cdot 2\cdot 5\cdot 3{,}14\cdot(6371+150)^2\cdot 10^6}{10^3}$$
$$=5341\ м$$

Формулу можно упростить:

$$\Delta h\simeq\frac{C_x\cdot\rho(h)\cdot 2\cdot S\cdot\pi\cdot {R_{зем}}^2}{m}$$

С учётом того, что плотность воздуха зависит от высоты линейно в заданном диапазоне высот, можно перейти к средней потере высоты за виток:

$$\Delta h_{ср}=\frac{5341+683}{2}=3000\ м$$

Тогда для снижения с высоты 200 км до (примерно) 155 км потребуется N витков:

$$N=\frac{200-155}{3}=15$$

Период обращения спутника определяется как N средних периодов обращения спутника

$$T=N\cdot\frac{2\cdot\pi\cdot(R_{зем}+h_{ср})}{V_{орб}}=15\cdot\frac{6{,}28\cdot(6371+175)\cdot 1000}{7784}=15\cdot 5281{,}2=79218\ с$$
$$=22\ часа$$

5) **(10 баллов)** Корпус спутника изготовлен из алюминия и имеет массу $m_к$=100 кг. Оцените величину нагрева корпуса спутника за 1 виток при характерной высоте орбиты H=150 км. Удельная теплоёмкость алюминия $c_{уд}$=897 Дж/(кг*К). Плотность воздуха на высоте 150 км составляет $2*10^{-9}$ кг/м^3. Принять, что на нагрев корпуса тратится половина теплоты, выделившейся при трении (остальное идёт на нагрев воздуха). Площадь поверхности корпуса спутника составляет $S_{пов}$ = 15 м^2, степень

черноты поверхности спутника ε=0,25. Считать, что изменение температуры за виток много меньше её абсолютной величины Т. Начальная температура спутника Т=300 К. Нагревом вследствие поглощения солнечного света пренебречь.

Указание:

- поток теплового излучения с единицы площади ($Вт/м^2$) описывается уравнением Стефана-Больцмана $Ф = \varepsilon \cdot \sigma \cdot T^4$ *,*
где $\sigma = 5{,}67 * 10^{-8}\ Вт/(м^2 * К^4)$*.*

-в расчёте потока теплового излучения для повышения точности использовать не начальную температуру Т, а среднюю температуру за виток (принять, что температура корпуса растёт линейно со временем).

-в вычислениях использовать упрощение для малых X: $(1+X)^n = 1+n*X$

Решение:

Работа сил трения за виток определена в п.4:

$$A_{трения} = F_{сопр} \cdot L \simeq -\frac{1}{2} \cdot C_x \cdot \rho \cdot S \cdot V_{орб}^2 \cdot 2 \cdot \pi \cdot (R_{зем} + H) =$$

$$\simeq -\frac{1}{2} \cdot C_x \cdot \rho \cdot S \cdot \frac{G \cdot M_{зем}}{R_{зем} + H} \cdot 2 \cdot \pi \cdot (R_{зем} + H) = \simeq -C_x \cdot \rho \cdot S \cdot G \cdot M_{зем} \cdot \pi =$$

$$= -2 \cdot (2 \cdot 10^{-9}) \cdot 5 \cdot (6{,}67 \cdot 10^{-11}) \cdot (5{,}97 \cdot 10^{24}) \cdot 3{,}14 = -25{,}0069 \cdot 10^6\ Дж$$

Определим количество теплоты, полученное корпусом спутника с учётом коэффициента 0,5:

$$Q_{получ} = \frac{1}{2} \cdot \left|A_{трения}\right|$$

Тогда имеем, что при периоде обращения t и средней температуре $T_{ср}$= (Т+0,5*ΔТ) отданное тепло (в пренебрежении изменением температуры):

$$Q_{отданное} = t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T_{ср}^4 = t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot \left(T + \frac{1}{2} \cdot \Delta T\right)^4$$

Период обращения t находим как:

$$t = \left[\frac{2\pi \cdot (R_{зем} + H)^{\frac{3}{2}}}{\sqrt{G \cdot M_{зем}}}\right] = \left[\frac{6{,}28 \cdot (6371 \cdot 1000 + 150 \cdot 1000)^{\frac{3}{2}}}{\sqrt{6{,}67 \cdot 10^{-11} \cdot 5{,}97 \cdot 10^{24}}}\right] = 5240\ с$$

Тогда можно определить нагрев корпуса ΔТ за один виток, составив уравнение:

$$c_{уд} \cdot m_к \cdot \Delta T = Q_{получ} - Q_{отданное}$$

$$c_{уд} \cdot m_к \cdot \Delta T = \frac{1}{2} \cdot \left|A_{трения}\right| - t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot \left(T + \frac{1}{2} \cdot \Delta T\right)^4$$

Учтём, что изменение температуры ΔТ<<Т по условию задачи

$$c_{уд} \cdot m_к \cdot \Delta T = \frac{1}{2} \cdot \left|A_{трения}\right| - t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^4 \cdot \left(1 + \frac{1}{2} \cdot \frac{\Delta T}{T}\right)^4$$

$$c_{уд} \cdot m_к \cdot \Delta T = \frac{1}{2} \cdot \left|A_{трения}\right| - t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^4 \cdot \left(1 + \frac{1}{2} \cdot 4 \cdot \frac{\Delta T}{T}\right)$$

Получаем линейное уравнение относительно неизвестной ΔT

$$c_{уд} \cdot m_к \cdot \Delta T = \frac{1}{2} \cdot \left|A_{трения}\right| - t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^4 - t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^3 \cdot 2 \cdot \Delta T$$

Переносим ΔT влево

$$\Delta T \cdot \left(c_{уд} \cdot m_к + 2 \cdot t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^3\right) = \frac{1}{2} \cdot \left|A_{трения}\right| - t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^4$$

Решаем относительно ΔT

$$\Delta T = \frac{\frac{1}{2} \cdot \left|A_{трения}\right| - t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^4}{c_{уд} \cdot m_к + 2 \cdot t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^3} = \frac{\frac{1}{2} \cdot \left|A_{трения}\right| - t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^4}{c_{уд} \cdot m_к + 2 \cdot t \cdot S_{пов} \cdot \varepsilon \cdot \sigma \cdot T^3} =$$

$$= \frac{\frac{1}{2} \cdot 25{,}0069 \cdot 10^6 - 5240 \cdot 15 \cdot 0{,}25 \cdot (5{,}67 \cdot 10^{-8}) \cdot 300^4}{897 \cdot 100 + 2 \cdot 5240 \cdot 15 \cdot 0{,}25 \cdot (5{,}67 \cdot 10^{-8}) \cdot 300^3} =$$

$$= \frac{12{,}5034 \cdot 10^6 - 9{,}0246 \cdot 10^6}{897 \cdot 100 + 60164{,}37} = 23\ K$$

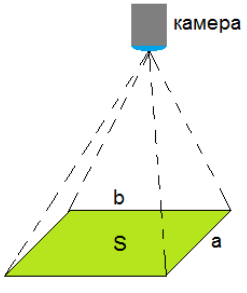
Задача 3 (спутниковая фотосъёмка)

Спутник движется по круговой орбите Земли, имеющей высоту h=500 км над уровнем моря. Масса Земли $5{,}97*10^{24}$ кг, а радиус 6371 км.

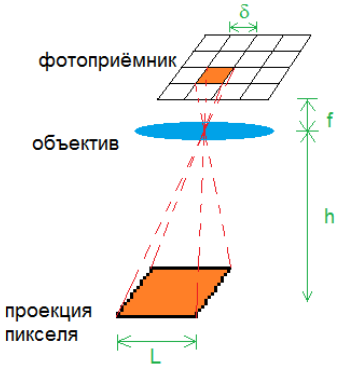
1) **(1 балл)** На спутнике размещена фотокамера с размером фотоприёмной матрицы 2500*3500 пикселей. Каким будет разрешение камеры, если она должна обеспечить съёмку площади 3500 Га за 1 кадр? Оптическая ось камеры направлена вертикально вниз.
   Решение:
   Найдём размер кадра (меньшую сторону прямоугольника *a*):

$$a = \sqrt{\frac{2500}{3500} \cdot S} = \sqrt{\frac{2500}{3500} \cdot 3500 \cdot 10^4} = 5000\ м$$

   Тогда разрешение камеры составляет L=5000 м /2500 пикселей = 2 метра



Задача 1



Задача 2

2) **(2 балла)** Какое фокусное расстояние должен иметь объектив камеры, чтобы обеспечить разрешение, определённое в п.1. при высоте орбиты h=500 км? Размер пикселя фотоприёмника δ=10 мкм.
Решение:
Из подобия треугольников имеем, что проекция пикселя L связана с размером пикселя фотоприёмника:

$$\frac{L}{\delta} = \frac{h}{f}$$

Тогда для фокусного расстояния f имеем:

$$f = \frac{\delta}{L} \cdot h = \frac{10^{-5}}{2} \cdot 500 \cdot 10^3 = 2{,}5\,м$$

3) **(2 балла)** Определите максимально допустимое время выдержки для указанных выше условий съёмки, чтобы изображение не было смазано. Оптическая ось объектива направлена вертикально вниз. Вращением Земли пренебречь.
Решение:
Скорость тени спутника по поверхности Земли (подспутниковой точки)

$$V_{пст} = \omega_{орб} \cdot R_{зем} = \frac{V_{орб}}{R_{зем} + h} \cdot R_{зем} = \sqrt{\frac{G \cdot M_{зем}}{(R_{зем} + h)^3}} \cdot R_{зем}$$

$$= \sqrt{\frac{6{,}67 \cdot 10^{-11} \cdot 5{,}97 \cdot 10^{24}}{(6371 + 500)^3 \cdot 10^9}} \cdot 6371 \cdot 10^3 = 7058{,}74\ \frac{м}{с}$$

Тогда скорость подспутниковой точки $V_{пст}$ и скорость бега изображения $V_{сби}$ в фокальной плоскости соотносятся как:

$$\frac{V_{сби}}{V_{пст}} = \frac{f}{h}$$

Из этого можно определить максимальное время выдержки: оно не должно превышать время смещения изображения на один пиксель.

$$T_{макс} = \frac{\delta}{V_{сби}} = \frac{\delta}{\frac{f}{h} \cdot V_{пст}} = \frac{10^{-5}}{\frac{2{,}5}{5 \cdot 10^5} \cdot 7058{,}74} = 2{,}83 \cdot 10^{-4} с$$

4) **(5 баллов)** Оцените время, за которое спутник отснимет всю поверхность Земли. Считать, что орбита полярная (проходит над полюсами Земли), а съёмка производится только на дневной стороне витка (при всех углах возвышения Солнца). Камера ориентирована вертикально вниз, причём длинная сторона кадра расположена поперёк направления полёта. Учесть, что после окончания съёмки не должно быть «разрывов» между снимками даже в самой широкой части Земли (для полярной орбиты): на Экваторе.
Решение:
Полоса захвата спутника составляет b = 3500*2 = 7000 метров
Тогда для гарантированного покрытия всей поверхности Земли нужно совершить N витков:

$$N = 2 \cdot \frac{2 \cdot \pi \cdot R_{зем}}{b} = 2 \cdot \frac{2 \cdot 3{,}14 \cdot 6371}{7} = 11431 \text{ витков}$$

Множитель 2 появляется, так как половина длины каждого витка нерабочая (теневая сторона Земли).

Период обращения спутника определяется как:

$$T_{sat} = \frac{2 \cdot \pi \cdot (R_{зем} + h)}{V_{орб}} = \frac{2 \cdot \pi \cdot (R_{зем} + h)}{\sqrt{\frac{G \cdot M_{зем}}{R_{зем} + h}}} = 2 \cdot \pi \cdot (R_{зем} + h) \cdot \sqrt{\frac{R_{зем} + h}{G \cdot M_{зем}}} =$$

$$= 6{,}28 \cdot (6371 + 500) \cdot 1000 \cdot \sqrt{\frac{(6371 + 500) \cdot 1000}{6{,}67 \cdot 10^{-11} \cdot 5{,}97 \cdot 10^{24}}} = 5668 \, c$$

Тогда оцениваем общее время съёмки Земли:

$$T_{all} = T_{sat} \cdot N = 5668 \cdot 11431 = 64{,}8 \cdot 10^6 c = 750 \text{ суток}$$

5) **(10 баллов)** Полярная орбита спутника такова, что, пролетая над Москвой, он движется на север. Во время пролёта над городом возникла срочная необходимость провести съёмку города Мурома. Естественно, что осуществить это можно только при помощи наклона линии визирования (съёмка с креном). Определите требуемый угол крена и разрешение полученного снимка Мурома по обеим координатам. Оба города находятся на одной широте 55,5 градусов. Долгота Москвы θ_{mos}=37,6841 градусов в.д., долгота Мурома θ_{mur}=42,05 градуса в.д. Принять, что расстояние между городами много меньше радиуса Земли.

Указание: проекция пикселя имеет вид не прямоугольника, а трапеции. В ответе приведите высоту трапеции и её среднюю линию.

Решение:

Радиус географической параллели 55,5 градусов R_{555}:

$$R_{555} = R_{зем} \cdot \cos 55{,}5^o = 3608{,}57 \text{ км}$$

Так как расстояние между городами много меньше радиуса Земли, то можно принять, что кратчайший путь совпадает с географической параллелью. Тогда расстояние между городами d определяется как:

$$d = R_{555} \cdot (\theta_{mur} - \theta_{mos}) = R_{зем} \cdot \cos 55{,}5^o \cdot (42{,}05 - 37{,}6841) \cdot \frac{3{,}14}{180} == 274{,}83 \text{ км}$$

При заданных условиях в расчёте крена Землю можно считать плоской. Тогда расстояние D от спутника до Мурома находим как:

$$D = \sqrt{h^2 + d^2} = \sqrt{500^2 + 275^2} = 570{,}63 \text{ км}$$

Отсюда угол между вертикалью и оптической осью:

$$\alpha = arccos\left(\frac{h}{D}\right) = 28{,}8 \text{ }^\circ$$

Находим разрешение по направлению, **совпадающему** с направлением полёта. Средняя линия трапеции (проекции пикселя) находится как:

$$l_{ср} = \frac{D}{f} \cdot \delta = \frac{\sqrt{h^2 + d^2}}{f} \cdot \delta = \frac{\sqrt{500^2 + 275^2}}{2{,}5 \cdot 10^{-3}} \cdot 10^{-5} = 2{,}28 \text{ м}$$

Теперь определим разрешение **поперёк** направления полёта. Это будет высота получившейся трапеции:

$$H_{тр} = \frac{\frac{D}{f} \cdot \delta}{cos\alpha} = \frac{\frac{D}{f} \cdot \delta}{\frac{h}{D}} = \frac{D^2}{fh} \cdot \delta = \frac{h^2 + d^2}{fh} \cdot \delta = \frac{500^2 + 275^2}{2{,}5 \cdot 10^{-3} \cdot 500} \cdot 10^{-5} = 2{,}61\ м$$

3.2. Задачи по информатике

Задача 3.2.1. 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.

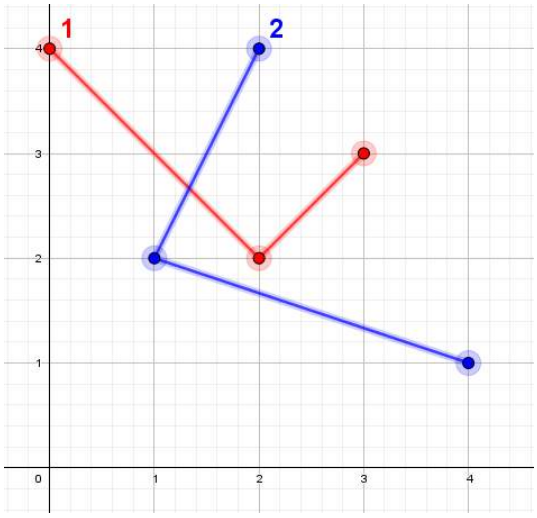


Рис. 1.1: Рисунок к первому примеру

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 3.2.2. 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

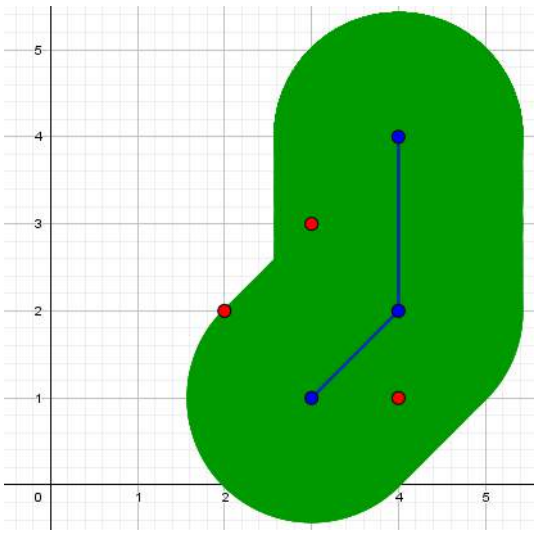


Рис. 1.2: Рисунок к первому примеру

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 3.2.3. 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.

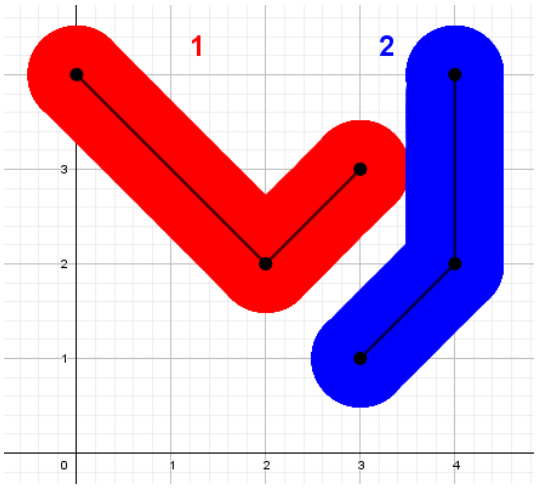


Рис. 1.3: Рисунок к первому примеру

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 3.2.4. 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.

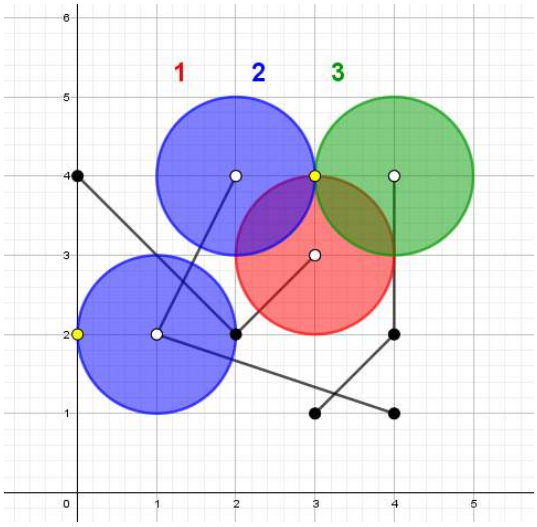


Рис. 1.4: Рисунок к первому примеру

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 3.2.5. 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 ($1 \leqslant i \leqslant n$; $1 \leqslant j \leqslant m$; $|x_{i,j}| \leqslant 10^9$; $|y_{i,j}| \leqslant 10^9$).

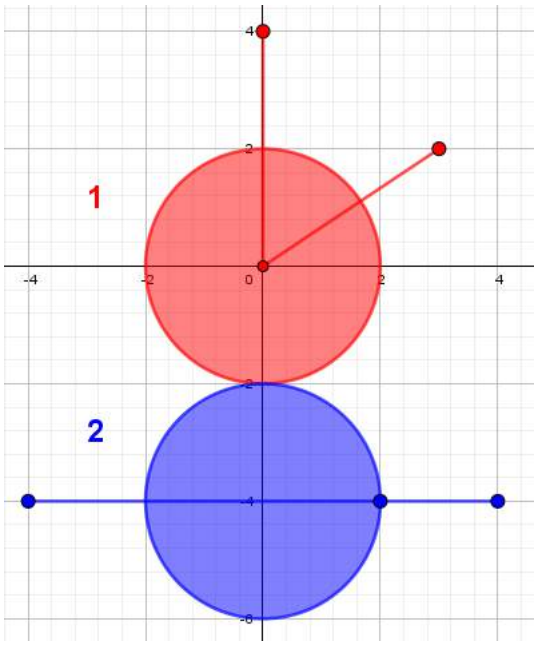


Рис. 1.5: Рисунок к первому примеру. Столкновение происходит примерно через 4 секунды после запуска.

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 3.2.6. 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

```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     }
```