2. ВТОРОЙ ОТБОРОЧНЫЙ ЭТАП

Второй отборочный этап проводится в командном формате в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Продолжительность второго этапа составляет 37 дней. Задачи условно разделены на задачи по математике и информатике, но носят междисциплинарный характер и в более простой форме воссоздают инженерную задачу заключительного этапа. Для каждой из параллелей (9 класс или 10-11 класс) предлагается свой набор задач по математике, задачи по информатике общие — для всех участников. Участники не были ограничены в выборе языка программирования для решения задач.

Объем и сложность задач этого этапа подобраны таким образом, чтобы решение всех задач одним человеком было маловероятно. Это призвано обеспечить включение командной работы и распределения обязанностей. Решение каждой задачи дает определенное количество баллов. Баллы зачисляются в полном объеме за правильное решение задачи. В данном этапе можно получить суммарно от 0 до 68 баллов.

Все условия задач по математике доступны участникам с первого дня второго отборочного этапа. Задачи по программированию выкладывались двумя партиями: в начале второго этапа и через три недели после начала. Команды могут выполнять задачи в любом порядке. Задачи допускают неограниченное число попыток сдать решение.

Задачи второго этапа

6.1. Задачи по математике (9 класс)

Задача 6.1.1. Многочлены (7 баллов)

Многочленом с одной переменной называется выражение вида

$$G(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \qquad (a_n \neq 0)$$

Числа $a_0, a_1, \ldots, a_n$ – это коэффициенты многочлена, a_n – старший коэффициент, a_0 – свободный член.

Степенью многочлена называют наибольшую степень переменной, входящую в многочлен и обозначают $\deg G(x)$.

Два многочлена называются **равными**, если равны все их коэффициенты. Многочлен равен нулю, если все его коэффициенты равны нулю. Степень нулевого многочлена не определена.

Число α является **корнем** многочлена $G(x)$, если $G(\alpha) = 0$.

Основная теорема арифметики для многочленов: для любых двух многочленов $F(x)$ и $G(x)$ существует единственная пара многочленов $P(x)$ (частное) и $Q(x)$ (остаток) такая, что $F(x) = G(x) \cdot P(x) + Q(x)$, причём степень остатка $Q(x)$ меньше степени делителя $G(x)$, или $Q(x)$ нулевой многочлен.

Теорема Безу: остаток от деления многочлена $F(x)$ на $(x - \alpha)$ равен $F(\alpha)$.

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Чтобы однозначно задать многочлен степени n, достаточно знать значения этого многочлена в n различных точках числовой прямой.

б) Если $\deg G(x) = n$ и $\deg F(x) = m$, то $\deg F(G(x)) = m \cdot n$.

в) Многочлен $x^{10} - x^5 + 1$ делится на многочлен $2x^2 - 2x + 2$ без остатка.

г) Если всякий корень многочлена $G(x)$ является корнем многочлена $F(x)$, то многочлен $F(x)$ делится на многочлен $G(x)$.

д) У многочлена $F(x)$ сумма коэффициентов при четных степенях равна сумме коэффициентов при нечетных степенях. Тогда таким же свойством обладают все многочлены $F(x) \cdot G(x)$ при любых значениях коэффициентов $G(x)$.

е) Известно, что все коэффициенты произведения многочленов $F(x)$ и $G(x)$ с целыми коэффициентами делятся на простое число p. Тогда все коэффициенты $F(x)$ или $G(x)$ делятся на p.

ж) Известно, что многочлен $F(x)$ принимает целые значения для всех целых x. Тогда все коэффициенты $F(x)$ — целые.

з) Многочлен $G(x)$ третьей степени принимает значения $G(-1) = 22$, $G(0) = 14$, $G(1) = -4$, $G(2) = 4$. Тогда обязательно $G(3) = 74$.

Решение

Объясним только пункты, утверждения в которых неверны. а) неверно, т.к. многочлен 1 степени уже по одной точке не восстановить (существуют бесконечное количество прямых, проходящих через заданную точку на плоскости). Пункт г) $G(x) = x^2$, $F(x) = x^2 - x$. $x^2 - x \neq x^2 \cdot P(x)$. Пункт ж) многочлен $\frac{1}{2}x(x+1)$ принимает целые значения при целых x, но коэффициенты не целые.

Ответ: 01101101

Задача 6.1.2. Кривые на плоскости (7 баллов)

Все точки плоскости, координаты $(x; y)$, которых являются решениями уравнения $f(x; y) = 0$ называются графиком уравнения $f(x; y) = 0$.

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Уравнение $x^2 + y^2 = 2x + 2y - 1$ задает общие точки прямой и окружности.

б) Уравнение $y = \sqrt{6 - x - x^2}$ задает на плоскости одну ветьв параболы.

в) Площадь фигуры, заданной неравенством $|2x-3y|+|3x+2y| \leqslant 13$, равна 26.

г) Периметр фигуры, заданной уравнением $|x+15y-88|+|9x+9y-99|=63$, равен 38.

д) Если точка с координатами $(x;y)$ движется на плоскости по правилу:

$$\begin{cases} x = 13\cos t; \\ y = 31\sin t, \end{cases}$$

то траектория ее движения описывает эллипс.

е) Графиком уравнения $\sqrt{x^2+y^2+6x+8y+25}+\sqrt{x^2+y^2-4x-16y+68}=13$ является отрезок.

ж) Графиком уравнения $|4x-3y+5|-|2x+3y-5|=10$ является параллелограмм.

з) Графиком уравнения $ax^2+bxy+cy^2=1$ при $a,c \neq 0$ и $b^2-4ac>0$ является гипербола.

Решение

Верные утверждения:

а) Уравнение $x^2+y^2=2x+2y-1$ задает окружность.

б) Уравнение $y=\sqrt{6-x-x^2}$ задает на плоскости полуокружность.

в) Площадь фигуры, заданной неравенством $|2x-3y|+|3x+2y| \leqslant 13$, равна 26.

г) Периметр фигуры, заданной уравнением $|x+15y-88|+|9x+9y-99|=63$, равен 36.

д) Если точка с координатами $(x;y)$ движется на плоскости по правилу:

$$\begin{cases} x = 13\cos t; \\ y = 31\sin t, \end{cases}$$

то траектория ее движения описывает эллипс.

е) Графиком уравнения $\sqrt{x^2+y^2+6x+8y+25}+\sqrt{x^2+y^2-4x-16y+68}=13$ является отрезок.

ж) Графиком уравнения $|4x-3y+5|-|2x+3y-5|=10$ является пара непересекающихся углов с соответственно параллельными сторонами.

з) Графиком уравнения $ax^2+bxy+cy^2=1$ при $a,c \neq 0$ и $b^2-4ac>0$ является гипербола.

Ответ: 00101101

Задача 6.1.3. ГМТ (6 баллов)

Геометрическим местом точек, обладающих некоторым свойством называется множество всех точек (плоскости), обладающих этим свойством.

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Даны точки A и B. ГМТ X таких, что $AX^2 + BX^2 = r^2$, где $r > AB$ является окружностью.

б) Даны точки A и B. ГМТ X таких, что $AX^2 - BX^2 = r^2$, где $r > 0$ является окружностью.

в) Даны точки A и B. ГМТ X таких, что $AX + BX = AB$, является прямой AB.

г) Существует единственная точка, равноудаленная от трех пересекающихся прямых.

д) Дан треугольник ABC. Внутри него взяли точку M и соединили ее с вершинами. Получилось три треугольника. ГМТ M, для которых сумма площадей двух из этих треугольников будет равна площади третьего, — средние линии треугольника.

е) Даны точки A и B. ГМТ X таких, что $\angle AXB = 36°$, является дугой окружности.

ж) Даны точки A и B. Множество всех точек C, таких, что ABC — равнобедренный треугольник, является прямой, перпендикулярной к AB.

з) На плоскости даны окружности S_1 и S_2. ГМТ центров окружностей, перпендикулярных S_1 и S_2 является радикальной осью этих окружностей.

Решение

Верные утверждения:

а) Не ограничивая общности введем систему координат так, чтобы точки A и B имели координаты $(-1; 0)$ и $(1; 0)$ соответственно. Пусть $X(x; y)$. Тогда уравнение $AX^2+BX^2 = r^2$, где $r > 2$, примет вид $(x+1)^2+y^2+(x-1)^2+y^2 = r^2$. Которое в свою очередь преобразуется к виду $x^2 + y^2 = \frac{r^2-2}{2}$.

б) В обозначениях пункта а) уравнение $AX^2 - BX^2 = r^2$ преобразуется к виду $4x = r^2$, что задает вертикальную прямую.

в) ГМТ X таких, что $AX + BX = AB$, является отрезок AB. Для остальных точек плоскости $AX + BX > AB$.

г) Существуют четыре точки, равноудаленных от трех пересекающихся прямых. Если взять треугольник, образованный этими прямыми, то искомые точки находятся в центрах вписанной и вневписанных окружностей этого треугольника.

д) Пусть $S_{AMB} = S_{AMC} + S_{BMC}$. Продолжим CM до пересечения с AB в точке D. Тогда $\dfrac{CM}{MD} = \dfrac{S_{AMC}}{S_{AMD}} = \dfrac{S_{BMC}}{S_{BMD}} = \dfrac{S_{AMC} + S_{BMC}}{S_{AMD} + S_{BMD}} = 1$.

е) ГМТ X таких, что $\angle AXB = 36°$, является объединением двух дуг окружности.

ж) Даны точки A и B. Множество всех точек C, таких, что ABC — равнобедренный треугольник, является объединением серединного перпендикуляра к AB и двух окружностей с центрами A и B и радиусом AB.

з) На плоскости даны окружности S_1 и S_2. ГМТ центров окружностей, перпендикулярных S_1 и S_2 является радикальной осью этих окружностей.

Ответ: 10001001

6.2. Задачи по математике (10-11 класс)

Задача 6.2.1. Интерполяция (7 баллов)

Задача состоит в том, чтобы неизвестную функцию, у которой известны значения в нескольких точках, приблизить полиномами.

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Даны $n+1$ точек $(x_i; y_i)$, $i = \overline{0,n}$, причем все x_i различны. Тогда существует единственный многочлен $P_n(x)$ степени n такой, что $P(x_i) = y_i$.

б) Интерполяционный многочлен второй степени, построенный по узлам $x_0 = 0, x_1 = 1, x_2 = 2$ для функции $f(x) = x^3$ в точке 3 дает погрешность в 6 единиц.

в) Определим функции

$$L_{n,k}(x) = \frac{(x-x_0)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_n)}{(x_k-x_0)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_n)}.$$

Тогда $L_{n,k}(x_k) = 1$ и $L_{n,k}(x_i) = 0$ при $i \neq k$.

г) Многочлен $P(x)$ 4 степени проходит через точки $(1;2), (2;1), (3;5), (4;6), (5;1)$. Тогда $P(0) = 21$.

д) Заданы центры $x_0 = 1$, $x_1 = 3$, $x_2 = 4$, $x_3 = 4,5$ и коэффициенты $a_0 = 5$, $a_1 = -2$, $a_2 = 0,5$, $a_3 = -0,1$, $a_4 = 0,003$. По ним вычислили интерполяционный многочлен Ньютона $P_4(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \dots + a_4(x-x_0)(x-x_1)(x-x_2)(x-x_3)$. Тогда $P_4(2,5) = 1,5$.

е) Пусть $T_n(x) = \cos(n \arccos(x))$ для $x \in [-1;1]$. Тогда при всех $n \in \mathbb{N}$ выполнятся соотношение:

$$T_{n+2}(x) = 2x^2 T_{n+1}(x) - T_n(x).$$

ж) Функция $T_n(x)$, определенная в предыдущем пункте, является четной функцией.

з) Функция $T_n(x)$ является многочленом степени n и имеет n различных корней на промежутке $[-1;1]$.

Решение

Объясним только пункты, утверждения в которых неверны.

а) Неверно, т.к. степень многочлена $P_n(x)$ может быть ниже n. Например, если все y_i равны между собой, то степень равна нулю.

д) $P_4(2,5) = 1,51$.

е) Верное соотношение:

$$T_{n+2}(x) = 2xT_{n+1}(x) - T_n(x).$$

ж) Функция $T_n(x)$ (см. многочлены Чебышева), является четной функцией при четных n и нечетной при нечетных n.

Ответ: 01110001

Задача 6.2.2. Параметризованная кривая (7 баллов)

Отображение $r(t) = (x(t); y(t))$, которое каждому значению t из интервала числовой прямой ставит в соответствие точку плоскости (или пространства) называется **параметризованной кривой**. Обычно параметр t определяет время, а точка $M(x(t); y(t))$ — материальную точку, которая движется на плоскости. Её вектор скорости в каждый момент времени находится по формуле $\vec{v}(t) = (x'(t); y'(t))$. Для определенности, время будем считать в секундах, а расстояние в метрах.

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Если точка с координатами $M(x; y)$ движется на плоскости по правилу:

$$\begin{cases} x = 3\cos t; \\ y = 3\sin t, \end{cases}$$

то траектория ее движения описывает окружность с радиусом 3.

б) Если точка с координатами $M(x; y)$ движется на плоскости по правилу:

$$\begin{cases} x = 2^t + 2^{-t}; \\ y = 2^t - 2^{-t}, \end{cases}$$

то эта точка движется по ветви гиперболы.

в) Скорость тела, движущегося на плоскости по закону

$$\begin{cases} x = t^3 - 3t; \\ y = t^2, \end{cases}$$

в момент времени $t = 2$ равна $\sqrt{97}$ (м/с).

г) Прямая $5x - 3y + 11 = 0$ касается кривой

$$\begin{cases} x = t^2 - t; \\ y = t^2 + t, \end{cases}$$

в точке $t = 2$.

д) Длина кривой $(x(t); y(t))$ в промежутке $a \leqslant t \leqslant b$ вычисляется по формуле:

$$\int_a^b \sqrt{(x'(t))^2 + (y'(t))^2}dt.$$

е) Площадь области, ограниченной кривой $(x(t); y(t))$, которую параметр $t \in [a; b]$ обходит против часовой стрелки, вычисляется по формуле

$$S = \int_a^b y(t)x'(t)dt.$$

ж) Тело движется на плоскости по правилу (единица измерения расстояния в метрах):

$$\begin{cases} x = 3\cos^3 \pi t; \\ y = 3\sin^3 \pi t. \end{cases}$$

Тогда каждые 2 секунды оно проходит по 18 метров.

з) Площадь области, ограниченной кривой

$$\begin{cases} x = t^3 - 4t; \\ y = t^2; \\ -2 \leqslant t \leqslant 2, \end{cases}$$

равна 15.

Решение

Объясним только пункты, утверждения в которых неверны. г) уравнение касательной в точке $t = 2$ имеет вид $5x - 3y + 8 = 0$. е) знак не правильный, эта формула считает площадь фигуры, которую параметр t обходит по часовой стрелке. з) $\int_{-2}^{2} t^2(3t^2 - 4)dt = \dfrac{256}{15}$.

Ответ: 11101010

Задача 6.2.3. ГМТ (6 баллов)

Геометрическим местом точек, обладающих некоторым свойством называется множество всех точек, обладающих этим свойством. (Например, ГМТ равноудаленных от данной точки называется сферой).

Про каждое утверждение ниже выясните: верно оно или нет. В ответ запишите строку, состоящую из восьми цифр 0 или 1, где 1 соответствует верному утверждению, а 0 — неверному. (*Например 01111001*).

а) Даны точки A и B. ГМТ X пространства таких, что $AX^2 + BX^2 = r^2$, где $r > AB$ является сферой.

б) Даны точки A и B. ГМТ X пространства таких, что $AX^2 - BX^2 = r^2$, где $r > 0$ является прямой.

в) Даны точки A и B. ГМТ X пространства таких, что $AX+BX = AB$, является прямой AB.

г) Существует единственная точка плоскости, равноудаленная от трех пересекающихся прямых в этой плоскости.

д) В пространстве проведены скрещивающиеся перпендикулярные прямые на расстоянии h друг от друга. Рассматриваются всевозможные отрезки длины d, один конец у которых на первой прямой, а второй — на другой. Тогда середины всех таких отрезков образуют окружность для всех $d > h$.

е) Даны точки A и B. ГМТ X таких, что $\angle AXB = 36^\circ$, является дугой окружности.

ж) На плоскости дан квадрат со стороной 1. Тогда объём тела, состоящего из ГМТ (пространства) на расстоянии 1 от этого квадрата равно $\frac{4\pi}{3} + 3$.

з) На плоскости даны окружности S_1 и S_2. ГМТ центров окружностей плоскости, перпендикулярных S_1 и S_2 является радикальной осью этих окружностей.

Решение

Верные утверждения:

а) Не ограничивая общности введем систему координат так, чтобы точки A и B имели координаты $(-1;0;0)$ и $(1;0;0)$ соответственно. Пусть $X(x;y;z)$. Тогда уравнение $AX^2 + BX^2 = r^2$, где $r > 2$, примет вид $(x+1)^2 + y^2 + z^2 + (x-1)^2 + y^2 + z^2 = r^2$. Которое в свою очередь преобразуется к виду $x^2 + y^2 + z^2 = \frac{r^2-2}{2}$, что является уравнением сферы.

б) В обозначениях пункта а) уравнение $AX^2 - BX^2 = r^2$ в координатах приводится к виду $x = \dfrac{r^2}{4}$, которое задает плоскость в пространстве.

в) ГМТ X пространства таких, что $AX + BX = AB$, является **отрезком** AB. Для точек вне отрезка AB выполняется $AX + BX > AB$.

г) Существуют четыре точки, равноудаленных от трех пересекающихся прямых. Если взять треугольник, образованный этими прямыми, то искомые точки находятся в центрах вписанной и вневписанных окружностей этого треугольника.

д) В пространстве проведены скрещивающиеся перпендикулярные прямые на расстоянии h друг от друга. Рассматриваются всевозможные отрезки длины d, один конец у которых на первой прямой, а второй — на другой. Тогда середины всех таких отрезков образуют окружность для всех $d > h$.

е) ГМТ X таких, что $\angle AXB = 36^\circ$, является поверхностью вращения дуги окружности вокруг отрезка AB.

ж) На плоскости дан квадрат со стороной 1. Тогда объём тела, состоящего из ГМТ (пространства) на расстоянии 1 от этого квадрата равно $\frac{4\pi}{3} + 2\pi + 2$.

з) На плоскости даны окружности S_1 и S_2. ГМТ центров окружностей плоскости, перпендикулярных S_1 и S_2 является радикальной осью этих окружностей.

Ответ: 10001001

6.3. Задачи по информатике

Задача 6.3.1. Одометрия (3 балла)

Робот, собранный по дифференциальной схеме, перемещается вдоль чёрной линии. Перемещение робота задаётся парами линейных ν и угловых ω скоростей. Пары скоростей передаются через равные промежутки времени t с. Значения скоростей не изменяются между замерами, они могут измениться мгновенно в момент проведения

измерения.

Робот начинает своё движение в точке $(0, 0)$ и имеет направление движения сонаправленное с направлением оси X. Необходимо определить расстояние от точки, где черная линия пересекает саму себя, до начала координат. В случае если пересечений несколько, то необходимо найти координату первого из них.

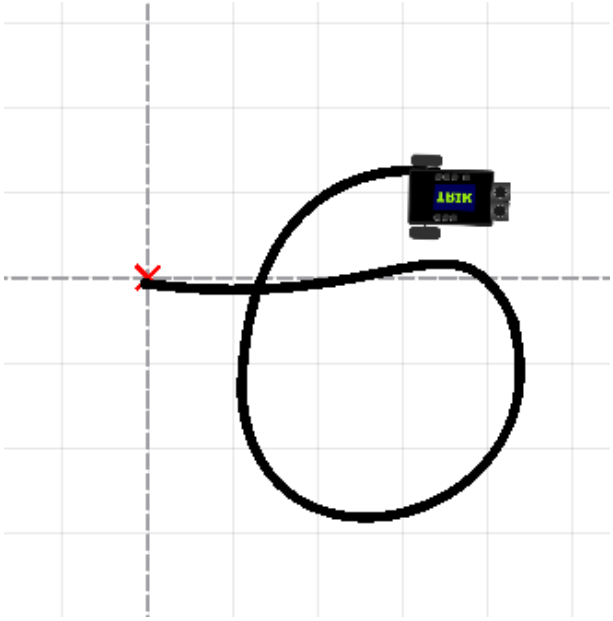


Рис. 6.1: Пример поля для тестирования робота

Формат входных данных

Первая строка входной файла содержит два числа N - количество замеров угловой и линейной скоростей ($0 \leqslant N \leqslant 10000$), и t - время между каждой парой замеров ($0 < t \leqslant 1$). N — целое число, t — вещественное число.

Далее идет N строк, в которых указана пара вещественных чисел ν мм/с и ω рад/с — линейная ($-1000 \leqslant \nu \leqslant 1000$) и угловая ($-100 \leqslant \omega \leqslant 100$) скорости через пробел .

Формат выходных данных

Расстояние в миллиметрах от начала координат до точки пересечения черной линии с самой собой. Допускается погрешность в 50 мм.

Примеры

Пример №1

Стандартный вход

```
33 0,5
133,902 0
137,323 -0,02
```

```
136,834 0
137,323 0,14
136,834 0,01
136,834 0,07
137,323 0,06
137,323 0,05
136,834 0
136,834 -0,2
137,323 -0,56
137,323 -0,66
136,834 -0,67
136,834 -0,65
137,323 -0,65
136,834 -0,57
137,323 -0,59
137,323 -0,51
136,834 -0,55
137,323 -0,58
136,834 -0,66
137,323 -0,69
136,834 -0,67
136,834 -0,56
137,323 -0,44
136,834 -0,35
137,323 -0,37
137,323 -0,34
136,834 -0,41
136,834 -0,47
137,323 -0,45
136,834 -0,45
137,323 -0,63
```

Стандартный выход

```
430
```

Решение

Рассмотрим движения робота между моментами изменения скоростей. В этот момент линейная и угловая скорость постоянны и мы можем рассматривать движение робота, как показано на рисунке (в начальный момент времени робот направлен вдоль оси X)

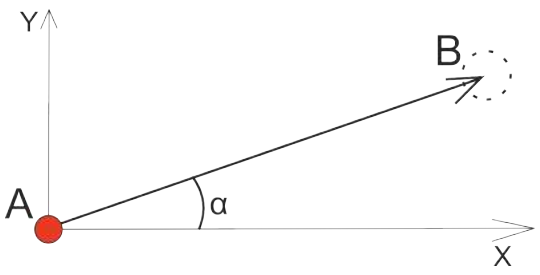


Рис. 6.2: Движение между измерениями скоростей из точки старта

Угол α мы можем вычислитть по формуле $\alpha = \omega t$. Длину отрезка AB можно высчитать по формул $l = \nu t$.

Далее мы узнаём координаты точки B по формуле $X_B = l\cos(\alpha)$ $Y_B = l\sin(\alpha)$

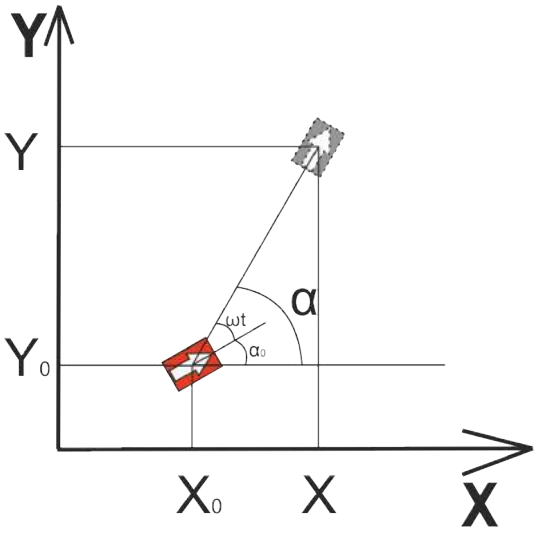


Рис. 6.3: Движение между измерениями скоростей

Теперь рассмотрим случай когда α_0 (начальный угол робота) $\neq 0$, то есть робот уже находится под каким то углов к оси X. В этом случае угол α будет вычисляться по формуле

$\alpha = \alpha_0 + \omega t$.

Дальше рассмотрим случай, когда наш робот уже находится в точке $(X_0; Y_0)$ при этом формула нахождения координат приобретёт вид

$X = X_0 + l\cos(\alpha)$

$Y = Y_0 + l\sin(\alpha)$

Используя данные формулы мы можем построить массив из отрезков

$(X_0; Y_0), (X_1; Y_1), (X_2; Y_2), (X_3; Y_3)...(X_N; Y_N)$,

Далее нам остаётся только проверить отрезки на пересечение друг с другом, для этого

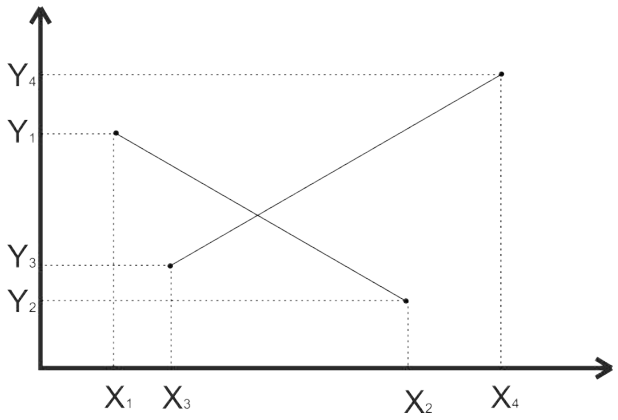


Рис. 6.4: Чертёж двух отрезков

Зная уравнения прямой:

$$y = kx + b$$

Найдём k и b для первого и второго отрезка

$$k = \frac{y_1 - y_2}{x_1 - x_2}$$

$$b = y - kx$$

(y_1 x_1 – координаты начала отрезка y_2 x_2 – координаты конца отрезка)

Далее найдём точку пересечения двух прямых

$$X = \frac{d_2 - b_1}{k_1 - k_2}$$

$$Y = k_1 x + b_1$$

Далее осталось проверить, лежит ли эта точка в пределах наших отрезков

$$X_{max} = Max(X_1, X_2, X_3, X_4)$$
$$Y_{min} = Min(Y_1, Y_2, Y_3, Y_4)$$

И если выполняется условие, то отрезки пересекаются

$$X < X_{max} and X > X_{min} and Y < Y_{max} and Y > Y_{min}$$

Пример программы

Ниже представлено решение на языке Java

```java
import java.awt.geom.Line2D;
import java.io.File;
import java.io.FileNotFoundException;
import java.math.RoundingMode;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Scanner;

public class Main {
    private static double life_cycle = 0;
    private static double time = 0;

    public static void main(String[] args) throws FileNotFoundException {

                // read input
        Scanner scan = new Scanner(System.in);
        ArrayList<Num> arrayOfSpeed = new ArrayList<>();
        ArrayList<Cordinate> arrayOfPoints = new ArrayList<>();
        life_cycle = Double.valueOf(scan.next().replace(',','.'));
        time = Double.valueOf(scan.next().replace(',','.'));

                //format control
        DecimalFormat df = new DecimalFormat("#.#");
        df.setRoundingMode(RoundingMode.HALF_DOWN);
        df.setMinimumFractionDigits(0);
```

```
26
27          int counter = 0;
28          arrayOfSpeed.add(new Num(0, 0));
29          while (counter != life_cycle) {
30              arrayOfSpeed.add(new Num(Double.valueOf(scan.next().
31                                  replace(',','.')), Double.valueOf(scan.next().
32                                  replace(',','.'))));
33              counter++;
34          }
35
36          arrayOfPoints.add(new Cordinate(0, 0));
37
38          double angle = 0;
39          double A1;
40          double A2;
41          double b1;
42          double b2;
43          double xIntersect = 0;
44          double yIntersect = 0;
45          double distance = 0;
46          boolean key = false;
47          int interCount = 0;
48
49          for (int i = 1; i < arrayOfSpeed.size(); i++) {
50              angle = angle + arrayOfSpeed.get(i).angleSpeed * time;
51              arrayOfPoints.add(
52                                  new Cordinate(arrayOfPoints.get(i - 1).x +
53                                  arrayOfSpeed.get(i).lineSpeed * time *
54                                  Math.cos(angle),arrayOfPoints.get(i - 1).y +
55                                  arrayOfSpeed.get(i).lineSpeed * time *
56                                  Math.sin(angle)));
57
58              if (arrayOfPoints.size() > 2) {
59                  for (int k = 1; k < arrayOfPoints.size() - 4; k++) {
60                      for (int j = k + 2; j < arrayOfPoints.size() - 1; j++) {
61                          key = Line2D.linesIntersect(
62                                                          arrayOfPoints.get(k).x,
63                                                          arrayOfPoints.get(k).y,
64                                                          arrayOfPoints.get(k + 1).x,
65                                                          arrayOfPoints.get(k + 1).y,
66                                                          arrayOfPoints.get(j).x,
67                                                          arrayOfPoints.get(j).y,
68                                                          arrayOfPoints.get(j + 1).x,
69                                                          arrayOfPoints.get(j + 1).y);
70                          if (key == true) {
71                              interCount = k;
72                              A1 = (arrayOfPoints.get(k).y -
73                                  arrayOfPoints.get(k + 1).y) /
74                                  (arrayOfPoints.get(k).x -
75                                  arrayOfPoints.get(k + 1).x);
76                              A2 = (arrayOfPoints.get(j).y -
77                                  arrayOfPoints.get(j + 1).y) /
78                                  (arrayOfPoints.get(j).x -
79                                  arrayOfPoints.get(j + 1).x);
80                              b1 = arrayOfPoints.get(k).y -
81                                  A1 * arrayOfPoints.get(k).x;
82                              b2 = arrayOfPoints.get(j).y -
83                                  A2 * arrayOfPoints.get(j).x;
84                              xIntersect = (b2 - b1) / (A1 - A2);
85                              yIntersect = xIntersect * A1 + b1;
```

```
86
87                              distance = Math.sqrt((xIntersect - 0) *
88                                  (xIntersect - 0) + (yIntersect - 0) *
89                                  (yIntersect - 0));
90                              System.out.println((int) (distance));
91                              return;
92                          }
93                      }
94                  }
95              }
96          }
97      }
98  }
99
100 class Num {
101     double lineSpeed;
102     double angleSpeed;
103
104     public Num(double lineSpeed, double angleSpeed) {
105         this.angleSpeed = angleSpeed;
106         this.lineSpeed = lineSpeed;
107     }
108
109     public double getLineSpeed() {
110         return lineSpeed;
111     }
112
113     public double getAngleSpeed() {
114         return angleSpeed;
115     }
116 }
117
118 class Cordinate {
119     double x;
120     double y;
121
122     public Cordinate(double x, double y) {
123         this.x = x;
124         this.y = y;
125     }
126
127     public double getX() {
128         return x;
129     }
130
131     public double getY() {
132         return y;
133     }
134 }
```

Задача 6.3.2. Построение карты (10 баллов)

На роботе, собранному по дифференциальной схеме, установлен инфракрасный дальномер так, что он направлен вправо относительно курса движения робота, передняя плоскость датчика совпадает с осью симметрии робота и выдвинута на Y мм относительно оси вращения в сторону передней части робота.

Он движется вдоль чёрной линии и его перемещение робота задаётся парами ли-

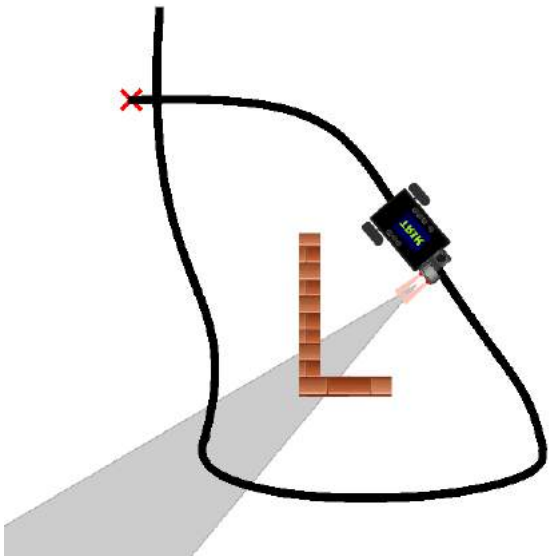


Рис. 6.5: Пример поля для входных данных №1

нейных ν и угловых ω скоростей. Пары скоростей и показания датчика передаются через равные промежутки времени t с. Диапазон работы датчика 0–100см. При этом если препятствие находится вне этого диапазона, то показания будут равны 100. Значения скоростей не изменяются между замерами, они могут измениться мгновенно в момент проведения измерения.

Робот начинает своё движение в точке $(0, 0)$ и имеет направление движения сонаправленное с направлением оси X. Необходимо определить является ли проекция препятствия замкнутым многоугольником, если известно, что на поле во время запуска робота точно не используются препятствия, чья площадь меньше Z мм^2.

Формат входных данных

Первая строка входной файла содержит четыре числа, разделённых пробелами: N — количество замеров угловой и линейной скоростей ($0 \leqslant N \leqslant 10000$), t — время между каждой парой замеров в секундах ($0 \leqslant t \leqslant 10000$), y — количество мм на сколько выдвинута передняя плоскость датчика расстояния относительно оси вращения ($0 \leqslant y \leqslant 10000$), и z — минимально возможная площадь препятствия в мм^2 ($0 \leqslant z \leqslant 100000$). N, y, z — целые числа, t — вещественное число.

Далее идет N строк, в которых указаны два вещественных числа и одно целое, разделённых пробелами: ν - линейная скорость ($-1000 \leqslant \nu \leqslant 1000$), ω - угловая скорость ($-100 \leqslant \omega \leqslant 100$), и d - показание дальномера ($0 \leqslant d \leqslant 100$).

Формат выходных данных

"True"или "False"в зависимости от того является или нет препятствие замкнутым многоугольником.

Примеры

Пример №1

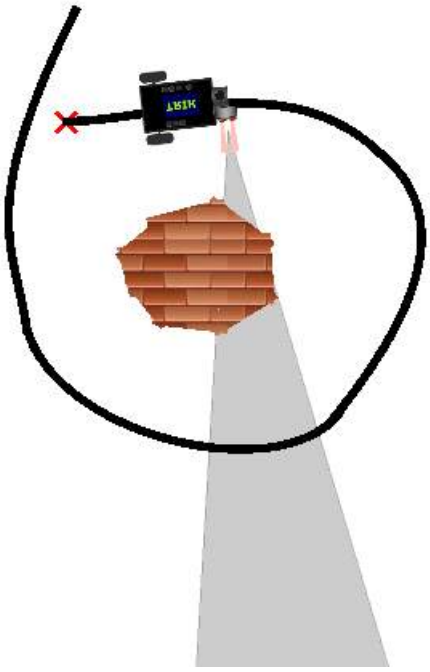


Рис. 6.6: Пример поля для входных данных №2

Стандартный вход

```
51 0,3 175 320523
202,807 0,12 100
221,54 -0,09 100
215,025 -0,13 100
190,59 -0,13 48
211,767 -0,15 23
199,549 -0,33 20
218,283 -0,41 17
209,323 -0,42 14
197,106 -0,5 17
209,323 -0,31 19
214,21 -0,33 22
211,767 -0,15 26
172,671 -0,04 28
216,654 -0,17 23
219,097 -0,07 18
223,169 -0,02 19
221,54 -0,02 100
221,54 -0,02 100
220,726 -0,1 100
215,025 -0,24 100
216,654 -0,3 100
180,002 -1,03 100
134,39 -1,63 42
175,115 -1,16 27
192,219 -0,74 22
183,26 -0,53 21
```

```
191,404 -0,4 20
196,291 -0,31 19
219,097 -0,34 18
208,509 -0,24 22
205,251 -0,2 49
201,178 -0,21 100
218,283 -0,35 100
196,291 -0,68 49
144,164 -1,54 23
182,445 -0,97 17
186,518 -0,44 15
219,097 -0,23 16
219,911 0,02 17
219,097 0,1 20
222,355 0,03 100
222,355 0,01 100
221,54 -0,06 100
222,355 -0,1 100
220,726 -0,16 100
217,468 -0,1 100
220,726 -0,1 100
214,21 -0,14 100
210,138 -0,19 100
218,283 -0,11 100
222,355 -0,01 100
```

Стандартный выход

```
False
```

Пример №2

Стандартный вход

```
47 0,3 175 117505
167,784 0,3 19
218,283 0,07 18
222,355 -0,01 17
217,468 0,01 16
219,911 -0,06 20
215,839 -0,12 24
216,654 -0,2 33
200,364 -0,24 100
207,694 -0,33 100
210,138 -0,41 100
204,436 -0,42 100
211,767 -0,32 100
210,138 -0,67 100
191,404 -0,66 100
197,106 -0,8 43
197,92 -0,72 29
196,291 -0,61 26
186,518 -0,38 27
```

```
209,323 -0,47 26
209,323 -0,36 27
205,251 -0,24 29
209,323 -0,38 32
191,404 -0,53 31
193,033 -0,57 29
193,848 -0,65 25
184,074 -0,45 24
210,138 -0,48 23
211,767 -0,5 20
197,92 -0,34 19
210,952 -0,38 22
215,839 -0,42 22
197,92 -0,25 25
181,631 -0,44 26
207,694 -0,46 24
210,952 -0,62 32
208,509 -0,67 28
209,323 -0,46 27
207,694 -0,34 100
209,323 -0,4 100
215,025 -0,35 100
217,468 -0,29 100
187,332 -0,34 100
209,323 -0,21 100
209,323 -0,14 100
219,097 -0,1 100
215,839 -0,18 100
216,654 -0,06 100
```

Стандартный выход

```
True
```

Решение

Рассмотрим движения робота между моментами изменения скоростей. В этот момент линейная и угловая скорость постоянны и мы можем рассматривать движение робота, как показано на рисунке (в начальный момент времени робот направлен вдоль оси X)

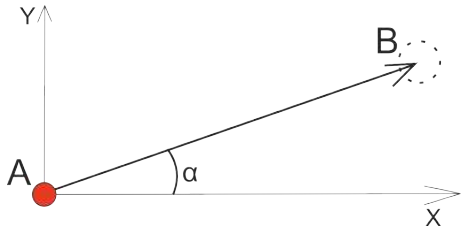


Рис. 6.7: Движение между измерениями скоростей из точки старта

Угол α мы можем вычислитть по формуле $\alpha = \omega t$. Длину отрезка AB можно высчитать по формул $l = \nu t$.

Далее мы узнаём координаты точки B по формуле $X_B = l\cos(\alpha)$ $Y_B = l\sin(\alpha)$

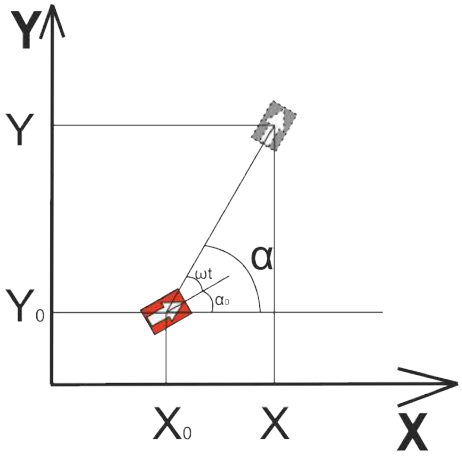


Рис. 6.8: Движение между измерениями скоростей

Теперь рассмотрим случай когда α_0 (начальный угол робота) $\neq 0$, то есть робот уже находится под каким то углов к оси X. В этом случае угол α будет вычисляться по формуле

$\alpha = \alpha_0 + \omega t.$

Дальше рассмотрим случай, когда наш робот уже находится в точке $(X_0; Y_0)$ при этом формула нахождения координат приобретёт вид

$X = X_0 + l\cos(\alpha)$

$Y = Y_0 + l\sin(\alpha)$

Далее нам нужно определить координаты препятствия

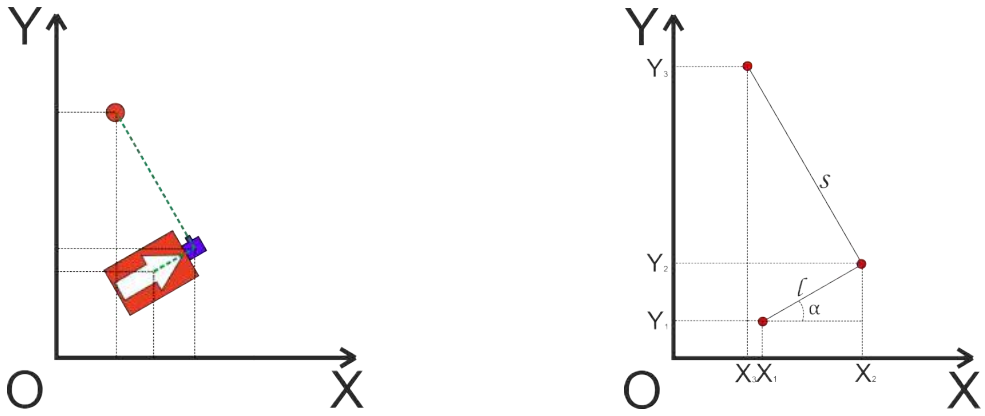


Рис. 6.9: Пересчёт координат робота в координаты препятствия

$$X_{obstacle} = X_{robot} + f\cos(\alpha) - s\sin(\alpha)$$
$$Y_{obstacle} = Y_{robot} + f\sin(\alpha) + s\cos(\alpha)$$

Далее находим площадь фигуры по формуле Гаусса

Где

- A - площадь многоугольника
- n — количество сторон многоугольника

$$\mathbf{A} = \frac{1}{2}\left|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n\right|$$

$$= \frac{1}{2}\left|x_1 y_2 + x_2 y_3 + \cdots + x_{n-1} y_n + x_n y_1 - x_2 y_1 - x_3 y_2 - \cdots - x_n y_{n-1} - x_1 y_n\right|$$

$$\mathbf{A} = \frac{1}{2}\left|\sum_{i=1}^{n} x_i (y_{i+1} - y_{i-1})\right| = \frac{1}{2}\left|\sum_{i=1}^{n} y_i (x_{i+1} - x_{i-1})\right| = \frac{1}{2}\left|\sum_{i=1}^{n} x_i y_{i+1} - x_{i+1} y_i\right| =$$

$$= \frac{1}{2}\left|\sum_{i=1}^{n} \det \begin{pmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{pmatrix}\right|$$

Рис. 6.10: Запись формулы Гаусса

- (x_i, y_i), i = 1, 2,..., n — координаты вершин многоугольника

Пример программы

Ниже представлено решение на языке Java

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.math.RoundingMode;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Scanner;

public class Main {
    private static double life_cycle;
    private static double time;
    private static double naScolko;
    private static double minS;

    public static void main(String[] args) throws IOException {

        Scanner scan = new Scanner(System.in);

        life_cycle = Double.valueOf(scan.next().replace(',','.'));
        time = Double.valueOf(scan.next().replace(',','.'));
        naScolko = Double.valueOf(scan.next().replace(',','.'));
        minS = Double.valueOf(scan.next().replace(',','.'));

        //создаем массивы координат и входных данных
        ArrayList<Cordinates> arrayOfCoordinates = new ArrayList<>();
        ArrayList<Nums> arrayOfInput = new ArrayList<>();
        ArrayList<Double> arrayOfDalnomer = new ArrayList<>();
        ArrayList<Double> arrayOf_L = new ArrayList<>();
        ArrayList<CoordinatesOfPolygon> arrayPointsPolygon =
        new ArrayList<>();
        ArrayList<Double> array = new ArrayList<>();
        ArrayList<Double> arrayCor = new ArrayList<>();

        //заполняем массив взодных данных
        int counter = 0;
        arrayOfInput.add(new Nums(0, 0, 0));
```

```
        while (counter != life_cycle) {
            arrayOfInput.add(new Nums(Double.valueOf(scan.next().
                replace(',', '.')), Double.valueOf(scan.next().replace(',', '.')),
            Double.valueOf(scan.next().replace(',', '.'))));
            counter++;
        }

        //добовляем координаты движения в массив кординат
        arrayOfCoordinates.add(new Cordinates(0, 0));
        double angle = 0;

        for (int i = 1; i < arrayOfInput.size(); i++) {

            angle = angle + arrayOfInput.get(i).angleSpeed * time;

            arrayOfCoordinates.add(new Cordinates(arrayOfCoordinates.get(i - 1).x +
                arrayOfInput.get(i).lineSpeed * time * Math.cos(angle),
                arrayOfCoordinates.get(i - 1).y + arrayOfInput.get(i).lineSpeed *
                time * Math.sin(angle)));
        }
                //отображаем координаты y
        for (int i = 0; i < arrayOfCoordinates.size(); i++) {
            arrayCor.add(i, arrayOfCoordinates.get(i).getY() * (-1));
        }

        angle = 0;
        for (int i = 0; i < arrayOfInput.size(); i++) {
            angle = angle + arrayOfInput.get(i).angleSpeed * time;
            array.add(angle);
        }

        //заполним массив видимых расстояний и где соточка(ничего не видим) ставим -1
        for (int i = 0; i < arrayOfInput.size(); i++) {
            if (arrayOfInput.get(i).dalnomer != 100) {
                arrayOfDalnomer.add(arrayOfInput.get(i).dalnomer);
            } else if (arrayOfInput.get(i).dalnomer == 100) {
                arrayOfDalnomer.add(-1.0);
            }
        }

        //получение и заполнение массива с координатами полигона которое обьезжаем
        //координаты полигона
        double xP;
        double yP;

        for (int i = 1; i < arrayOfDalnomer.size(); i++) {

            if (arrayOfDalnomer.get(i) != -1.0) {

                xP = arrayOfCoordinates.get(i).getX() + naScolko *
                    Math.cos(array.get(i)) + Math.sin(array.get(i)) *
                    arrayOfDalnomer.get(i) * 10;

                yP = arrayCor.get(i) - naScolko * Math.sin(array.get(i)) +
                    Math.cos(array.get(i)) * arrayOfDalnomer.get(i) * 10;

                arrayPointsPolygon.add(new CoordinatesOfPolygon(xP, yP));

            }

```

```
        }

                //используем формулу Гаусса для подсчёта площади
        double s = 0, s1 = 0;
        for (int j = 0; j < arrayPointsPolygon.size() - 1; j++) {
            s = s + (arrayPointsPolygon.get(j).getX() *
                arrayPointsPolygon.get(j + 1).getY());
            s1 = s1 + (arrayPointsPolygon.get(j).getY() *
                arrayPointsPolygon.get(j + 1).getX());
        }
        s = s + arrayPointsPolygon.get(arrayPointsPolygon.size() - 1).getX() *
            arrayPointsPolygon.get(0).y;
        s1 = s1 + arrayPointsPolygon.get(arrayPointsPolygon.size() - 1).getY() *
            arrayPointsPolygon.get(0).x;

        DecimalFormat form = new DecimalFormat("#.##");
        form.setRoundingMode(RoundingMode.HALF_DOWN);
        form.setMinimumFractionDigits(2);
        double space = (s1 - s) / 2;

                //проверка площади на знак
        if (space < 0) {
            space = space * (-1);
            if (space >= minS) {
                System.out.println("True");
            } else if (space < minS) {
                System.out.println("False");
            }
        } else if (space > 0) {
            if (space >= minS) {
                System.out.println("True");
            } else if (space < minS) {
                System.out.println("False");
            }
        }
    }
}
class CoordinatesOfPolygon {
    double x;
    double y;

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

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }
}
class Cordinates {
    double x;
    double y;


    public Cordinates(double x, double y) {
```

```
157         this.x = x;
158         this.y = y;
159     }
160 
161     public double getX() {
162         return x;
163     }
164 
165     public double getY() {
166         return y;
167     }
168 
169 }
170 class Dalnomer {
171     double dalnomer;
172 
173     public void Dalnomer(double dalnomer) {
174         this.dalnomer = dalnomer;
175     }
176 
177     public double getDalnomer() {
178         return dalnomer;
179     }
180 }
181 class Nums {
182     double lineSpeed;
183     double angleSpeed;
184     double dalnomer;
185 
186     public Nums(double lineSpeed, double angleSpeed, double dalnomer) {
187         this.angleSpeed = angleSpeed;
188         this.lineSpeed = lineSpeed;
189         this.dalnomer = dalnomer;
190     }
191 
192     public double getLineSpeed() {
193         return lineSpeed;
194     }
195 
196     public double getAngleSpeed() {
197         return angleSpeed;
198     }
199 
200     public double getDalnomer() {
201         return dalnomer;
202     }
203 }
204 
205 class Smen {
206     double hy;
207     double lx;
208 
209     public Smen(double hy, double lx) {
210         this.hy = hy;
211         this.lx = lx;
212     }
213 
214     public double getHy() {
215         return hy;
216     }
```

```
217
218     public double getLx() {
219         return lx;
220     }
221 }
```

Задача 6.3.3. Исследование лабиринта (5 баллов)

Робот, собранный по дифференциальной схеме, оснащен тремя инфракрасными датчиками расстояния.Один из датчиков расстояния установлен так, что показывает расстояние до препятствий прямо по курсу робота. Второй датчик расстояния направлен влево. Последний — вправо.

Показания датчиков расстояния бинарные, т.е.

- 1 — данный датчик не увидел препятствие, проезд в данном направлении свободен
- 0 — данные датчик увидел препятствие, проезд в данном направлении перекрыт

Первоначальная структура лабиринта известна в виде списка достижимости секций лабиринта. Связь в списке двунаправленная. Первоначальный сектор старта и направление движения также известны.

Считается, что непосредственно перед стартом робота случилось обрушение каких-то секций лабиринта. Поэтому роботу необходимо в процессе обхода всего лабиринта, выяснить какие секции оказались недоступны в результате обрушения.

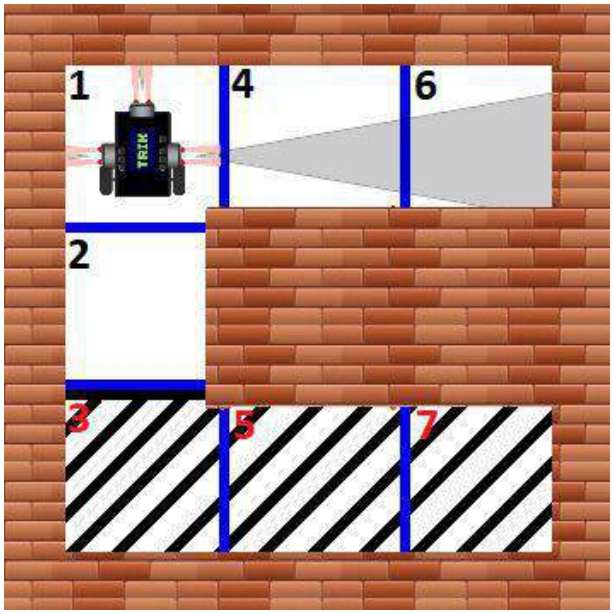


Рис. 6.11: Пример изображения для входных данных

Формат входных данных

Первая стока входного файла содержит три целых числа, разделённые пробелами: M — общее количество секций в лабиринте ($0 \leqslant M \leqslant 10000$), N — количество

соединений комнат ($0 \leqslant N \leqslant 10000$), и K — количество замеров ($0 \leqslant K \leqslant 10000$), произведенных датчиками, в процессе обхода всего лабиринта.

Далее идет N строк, в которых указаны 3 целых числа — номера секторов, между которые связаны между собой и направление связи 0–3 (0—вверх, 1—вправо, 2—вниз, 3—налево).

Потом идет строчка содержащая 2 целых числа: сектор и направление старта.

После указаны K строк, в которых указаны 3 цифры: первая — показание инфракрасного датчика, направленного влево относительно курса робота, вторая — показание инфракрасного датчика, направленного прямо по курсу робота, третья — показание инфракрасного датчика, направленного вправо относительно курса робота. При этом если в строке все числа 3, то это означает поворот робота вокруг своей оси по часовой стрелке на 90°, если 2 — против часовой стрелки.

Формат выходных данных

Вывести на экран номера всех секций, которые оказались недоступны, через пробел, в порядке возрастания их номеров.

Примеры

Пример №1

Стандартный вход

```
7 6 18
4 1 3
2 1 0
3 2 0
5 3 3
6 4 3
7 5 3
1 0
0 0 1
3 3 3
0 1 1
0 1 0
0 0 0
2 2 2
1 0 0
2 2 2
0 1 0
0 1 0
1 0 0
2 2 2
1 1 0
0 0 0
2 2 2
1 0 0
2 2 2
0 1 0
```

Стандартный выход

```
3 5 7
```

Комментарии

На рисунке 6.15 перечёрнкутые сектора — недоступные сектора. Показания датчиков: 0 0 1. Нумерация секторов также представлена на рисунке.

Решение

По условию задачи, мы знаем, что роботу для составления полной картины лабиринта нужно будет проехать все доступные ему участки. По этому опираясь на начальную позицию и начальное направление робота, мы можем построить его маршрут. При этом зная предыдущий сектор и направление проезда робота, определить тот сектор в котором робот окажется после своего проезда вперёд, не составит труда (в условии задаче дан список всех переходов с их направлениями). Изменение направления же явно указано во входных данных. В конце этого алгоритма мы получаем список комнат в которых побывал робот, вычтя их из списка всех комнат мы получим ответ.

Пример программы

Ниже представлено решение на языке Python3

```python
def turnRight(initialPosition):
    return (initialPosition + 1) % 4

def turnLeft(initialPosition):
    return (initialPosition - 1) % 4

def symmetricAdd(room, a, b, c):
    if c == 3:
        room[1][b - 1] = a
    elif c == 2:
        room[0][b - 1] = a
    elif c == 1:
        room[3][b - 1] = a
    else:
        room[2][b - 1] = a

n, k, l = map(int, input().strip().split())
rooms = []
for i in range(4):
    rooms.append([-1 for j in range(n)])
for i in range(k):
    a, b, c = map(int, input().strip().split())
    rooms[c][a-1] = b
    symmetricAdd(rooms, a, b, c)
initialPosition, initialDirection = map(int, input().strip().split())
newRooms = []
for i in range(4):
    newRooms.append([-1 for j in range(n)])

```

```
currentRoom = initialPosition
currentDirection = initialDirection
turnFlag = True

for i in range(l):
    left, middle, right = map(int, input().strip().split())
    if left == middle == right == 3:
        currentDirection = turnRight(currentDirection)
        turnFlag = True
    elif left == middle == right == 2:
        currentDirection = turnLeft(currentDirection)
        turnFlag = True
    else:
        if not turnFlag:
            currentRoom = newRooms[currentDirection][currentRoom - 1]
        if left == 1:
            newRooms[turnLeft(currentDirection)][currentRoom-1] = \
                rooms[turnLeft(currentDirection)][currentRoom-1]
            symmetricAdd(newRooms, currentRoom,
                    rooms[turnLeft(currentDirection)][currentRoom-1],
                         turnLeft(currentDirection))
        if right == 1:
            newRooms[turnRight(currentDirection)][currentRoom - 1] = \
                rooms[turnRight(currentDirection)][currentRoom - 1]
            symmetricAdd(newRooms, currentRoom,
                         rooms[turnRight(currentDirection)][currentRoom - 1],
                         turnRight(currentDirection))
        if middle == 1:
            newRooms[currentDirection][currentRoom - 1] = \
                rooms[currentDirection][currentRoom - 1]
            symmetricAdd(newRooms, currentRoom,
                    rooms[currentDirection][currentRoom - 1],
                    currentDirection)
        turnFlag = False

allRooms = set()
allNewRooms = set()
for i in range(4):
    for j in newRooms[i]:
        if j != -1:
            allNewRooms.add(j)
for i in range(4):
    for j in rooms[i]:
        if j != -1:
            allRooms.add(j)

hiddenRooms = allRooms.difference(allNewRooms)
sortedHiddenRooms = sorted(list(hiddenRooms))

for i in sortedHiddenRooms:
    print(i, end = " ")
```

Задача 6.3.4. Оптическая одометрия (5 баллов)

Робот, собранный по дифференциальной схеме, оборудован камерой и движется прямо вперёд.

Камера на роботе установлена так, что смотрит вниз перпендикулярно поверхности, по которой движется робот, известны характеристики камеры такие как раз-

решение ($M \times N$), угол обзора α, высота расположения над плоскостью движения. Верхний край кадра находится дальше от робота, чем нижний край кадра. Есть последовательность кадров, сделанных камерой через равные промежутки времени. Время между кадрами равно 1 секунде. Зная что робот движется только прямо и скорость его движения постоянна, определить расстояние на которое переместился робот.

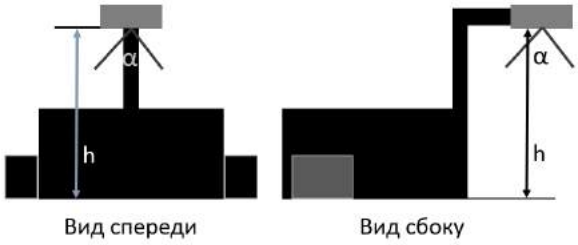


Рис. 6.12: Вид робота. Поясняющая картинка

Формат входных данных

Первая строка входных данных содержит 4 числа, целочисленные M, N и вещественные α, h — разрешение фотографии в пикселях, где M - длина, N - ширина ($2 \leqslant M, N \leqslant 60$), угол обзора камеры в градусах ($10° \leqslant \alpha \leqslant 170°$), высота расположения камеры над плоскостью движения в сантиметрах ($1 \leqslant h \leqslant 100$). Во второй строке задано целое число K — количество снимков ($2 \leqslant K \leqslant 8$). В следующих $K * M$ строках перечисляются N целых чисел — 0 или 1.

Формат выходных данных

Выведите единственное число — расстояние в см, на которое переместился робот с точностью до 2 символов.

Примеры

Пример №1

Стандартный вход

```
8 8 54 14
3
1 1 1 0 0 1 1 0
0 0 1 1 0 0 1 0
1 0 0 1 0 1 1 0
0 0 1 1 1 0 0 1
0 1 0 1 0 1 1 0
1 0 1 1 0 0 0 1
0 0 1 0 0 1 0 1
1 0 0 1 1 1 0 1
1 0 1 1 0 0 0 1
0 0 1 0 0 1 0 1
```

```
1 0 0 1 1 1 0 1
0 1 1 0 1 1 0 1
0 0 0 1 1 0 0 1
0 0 1 1 1 1 0 0
1 1 1 1 1 1 0 0
0 0 1 1 0 0 1 0
0 0 1 1 1 1 0 0
1 1 1 1 1 1 0 0
0 0 1 1 0 0 1 0
1 0 0 1 1 1 0 1
0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 0
0 0 0 0 1 1 0 1
0 1 1 0 0 1 0 1
```

Стандартный выход

```
26.75
```

Решение

Для лучшего понимания представим нашу камеру в виде равнобедренного треугольника с опущенной высотой

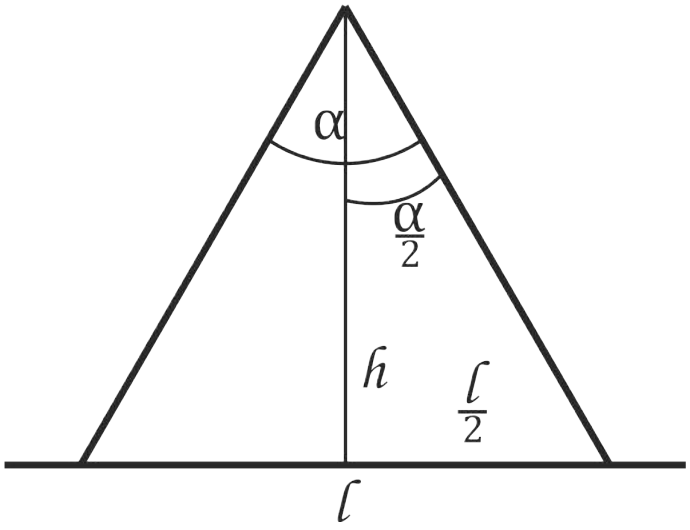


Рис. 6.13: Геометрическое представление камеры робота

Зная высоту h и угол α мы можем найти l по формуле $l = 2h\,\mathrm{tg}(\frac{\alpha}{2})$

Далее мы можем найти длину одного пикселя $l_p = \frac{l}{M} = \frac{2h\,\mathrm{tg}(\frac{\alpha}{2})}{M}$

Смещение камеры в пикселях(r) находится алгоритмом построчного сравнения кадров.

Далее подставляем в формулу и находи смещение в см $l_s = \frac{2rh\,\mathrm{tg}(\frac{\alpha}{2})}{M}$

Так как интервал между кадрами 1сек скорость в см/с равна смещению в см. А пройденный путь $S = \nu t$

$$S = \frac{2Krh\,\mathrm{tg}(\frac{\alpha}{2})}{M}$$

Пример программы

Ниже представлено решение на языке Python3

```
import math
data = input().split()

m = int(data[0])
n = int(data[1])
a = float(data[2])
h = float(data[3])
countFrame = int(input())

file = ''
for i in range(countFrame*m):
    file +=input() +'\n'

lines = file.split('\n')

res = -1
nRes = 0
for r in range(countFrame - 1):
    if res != -1: break
    nRes = 0
    for i in range(m):
        if lines[i + m*r ].split() == lines[ m + m*r].split():
            c = 0
            for j in range(m - i):
                if lines[ i + m*r +j ].split() == lines[m + m*r + j]\
                        .split():
                    c = j + 1
                else:break
            if c == m - i:
                res = i
                nRes += 1
        if nRes > 1: res = -1;
print(round(2*math.tan(a*math.pi/180/2)*h*res/m*countFrame,2))
```

Задача 6.3.5. Фильтрация значений (5 баллов)

Робот, собранный по дифференциальной схеме, имеет инфракрасный дальномер и перемещается прямо по одной прямой. Дальномер установлен так, что он направлен влево относительно курса движения робота.

Необходимо определить форму исследуемой стены. Она может одной из следующих:

- линейная (linear)
- модуль (module)
- квадратичная (quadratic)
- кубическая (cubic)
- синусоида (sinus)

Показания датчика не идеальные, существует некоторая погрешность измерений. Гарантируется, что стена всегда находится в поле зрения датчика.

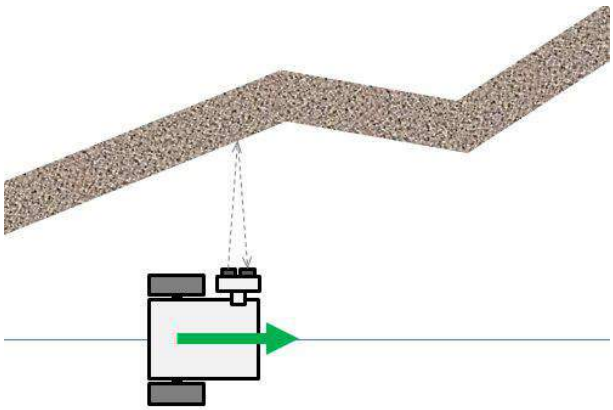


Рис. 6.14: Движение робота относительно стены

Формат входных данных

Первая строка входного файла содержит одно целое число: N — количество ($200 \leqslant N \leqslant 10000$) показаний датчика, снятых в процессе движения вдоль стены.

Последующие N строк содержат одно целое число — s — показание дальномера ($0 \leqslant s \leqslant 1200$).

Формат выходных данных

Одна стока, содержащая одно слово - форму стены. Слово должно быть на английском одним из списка, представленного при описании формы стены.

Примеры

Пример №1

Стандартный вход

```
206
13
17
17
18
18
21
21
22
23
22
22
21
26
25
26
27
31
```

32
30
35
33
35
37
37
40
41
41
44
42
45
46
46
49
50
48
50
52
53
53
54
60
58
61
61
61
62
63
61
65
63
65
66
66
67
70
73
72
71
74
75
76
77
78
82
79
81
79
81
80
83

82
84
83
83
82
81
79
79
79
80
77
77
78
74
74
73
74
71
72
71
70
67
66
66
63
61
62
59
62
57
56
54
54
49
48
48
50
49
45
45
43
45
41
41
37
39
37
35
36
37
36
34
35

33
32
30
32
33
28
30
30
31
28
30
31
29
31
28
27
30
29
29
27
29
29
27
28
29
29
31
31
29
30
29
30
30
31
33
33
34
33
33
38
33
37
36
37
35
36
39
39
36
40
42
44
43

```
46
47
49
49
51
53
55
54
56
59
60
59
65
62
63
66
67
67
69
69
72
77
76
79
82
83
83
82
85
84
```

Стандартный выход

```
cubic
```

Комментарии

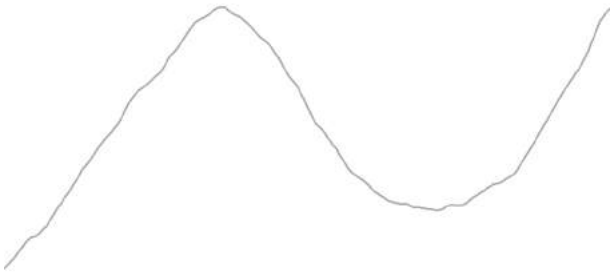


Рис. 6.15: Пример графика по входным данным

На рис. 6.15 изображен график, построенный по входным значениям с некоторой фильтрацией. По графику видно, что исследуемая стена имеет вид кубической функции, следовательно необходимо вывести ”cubic”

Рекомендуется использовать теорему Лагранжа (http://www.math24.ru/%D1%82%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0-%D0%BB%D0%B0%D0%B3%D1%80%D0%B0%D0%BD%D0%B6%D0%B0.html)

Решение

Считаем значения и обработаем их медиальным и сглаживающим фильтрам.

Далее мы можем заметить, что у разных функций разное количество точек перегиба (линейная – 0, модуль и квадратичная – 1, кубическая – 2, синусоида – 3 и больше).

Точки перегиба – точки, где функция меняет свой характер (возрастающий переходит убывающий и наоборот).

Возрастающая функция – функция в которой выполняется условие, что каждое последующее значение больше предыдущего ($x_i > x_{i-1}$)

Убывающая функция – функция в которой выполняется условие, что каждое последующее значение меньше предыдущего ($x_i < x_{i-1}$)

Зная это мы можем определять количество точек перегиба, для этого достаточно найти разницу между x_i и x_{i-1} (если она положительная то функция на этом участке имеет возрастающий характер, если нет, то убывающий), если $x_i - x_{i-1}$ и $x_{i+1} - x_i$ имеют разные знаки, то x_i это точка перегиба.

Для того что бы мы могли различить функцию модуля и квадратичную функцию, давайте рассмотрим их внимательнее.

Квадратичная функция

Функция модуля

Рис. 6.16: Сравнение функций

Как мы видим, функция модуль всегда возрастает и убывает с одной той же скоростью, а квадратичная с разной, чем дальше от точки перегиба тем больше эта скорость. То есть $|x_i - x_{i-1}| = |x_{i+1} - x_i|$ то перед нами функция модуля, а если $|x_i - x_{i-1}| \neq |x_{i+1} - x_i|$, то квадратичная. Хочу заметить, что в нашем случае строгое равенство не подходит, так как мы имеем дело с неидеальными данными (данные, на которых присутствует шум, который мы хотя и практически полностью убрали фильтрами, но его следы всё таки присутствуют в данных), по этому в программе допускаем небольшую погрешность в моменте сравнения $|x_i - x_{i-1}|$ и $|x_{i+1} - x_i|$.

Пример программы

Ниже представлено решение на языке Python3

```
1 dataset = []
2 dataset.append(input())
3
```

```
for i in range(int(dataset[0])):
    dataset.append(input())

k = 4
d = 3

line = []

for i in range(int(dataset[0])//d):
    val = int(dataset [1+i*d])
    for j in range(d):
        if val < int(dataset[1+i*d+j]):
            val = int(dataset[1+i*d + j])
    line.append(val)

for i in range(1,int(dataset[0])//d):
    line[i]= (line[i-1]*k+line[i])/(k+1)

dif = []
for i in range(5,int(dataset[0])//d):
    dif.append(line[i]-line[i-1])

c = 0

for i in range(2,len(dif)):
    if dif[i-2]*dif[i-1] < 0 and dif[i-2]*dif[i]:
        c+=1

if c == 0:
    print('linear')
elif c == 1:
    if abs(dif[len(dif)//4*3] - dif[len(dif)-3])>1:
        print('quadratic')
    else: print('module')
elif c == 2:
    print('cubic')
else:
    print('sinus')
```

Задача 6.3.6. Всенаправленная тележка (5 баллов)

Робот, моторы которого расположены под углом в 120° друг к другу, движется в заданном направлении некоторое время t. На валах моторов закреплены омниколёса (`https://en.wikipedia.org/wiki/Omni_wheel`).

Необходимо определить новые координаты центра робота, если в момент начала движения он находился в точке $(0, 0)$ и один из двигателей находился на оси Y, в направлении положительной части (см. рис. 6.17). Расположение моторов является постоянным и соответствует расположению моторов на рисунке 6.17.

Считать что, мощность на все моторы подаётся одновременно, время их вращения одинаковое, при этом они достигают своей скорости мгновенно, колеса вращаются без проскальзывания.

Формат входных данных

Одна строчка, состоящая из 5ти чисел: d, w_1, w_2, w_3, t, где

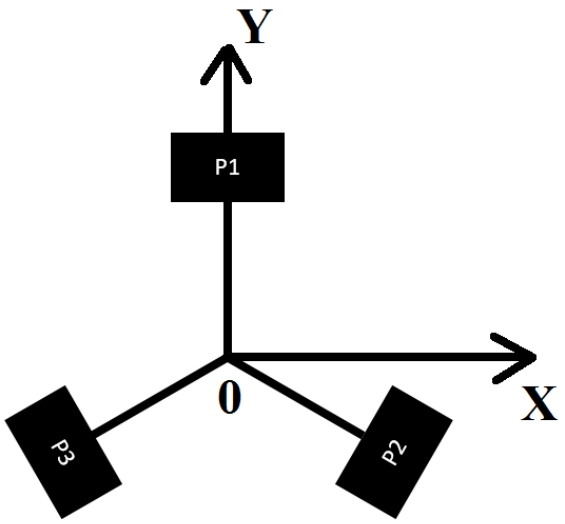


Рис. 6.17: Расположение моторов робота относительно центра глобальной системы отсчета в начальный момент времени

- d — диаметра колёс в мм ($d_1, d_2, d_3 = d; 30 \leqslant d \leqslant 100$);
- $P1, P2, P3$ — скорости вращения моторов (-2 рад/с $\leqslant P1, P2, P3 \leqslant 2$ рад/с);
- t — времени движения в с ($10 \leqslant t \leqslant 1000$).

Диаметр колёс и время движения — целые числа, скорости вращения — вещественные.

Формат выходных данных

Одна строка, содержащая два вещественных числа с точностью до одного знака — координаты робота, где он закончил своё движение, через пробел. Числа указывать с точностью до сотых. Допускается погрешность в 1 мм по каждой из координат.

Примеры

Пример №1

Стандартный вход

```
30 1.5 -1.0 -0.5 10
```

Стандартный выход

```
337.5 64.95
```

Пример №2

Стандартный вход

```
100 -0.29 -0.3 0.59 10
```

Стандартный выход

```
-217.5 385.37
```

Решение

Рассмотрим модель, у нас есть 3 жёстко соединённые точки которые вращаются под углом 120°

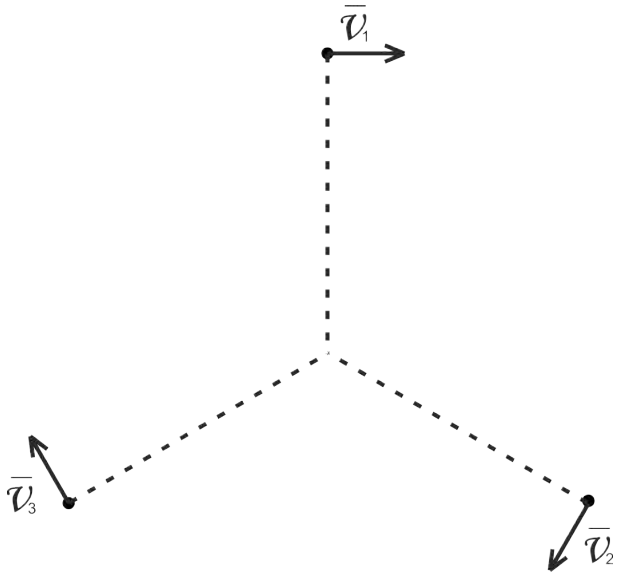


Рис. 6.18:

как можно заметить, что при при скоростях не выполняющих условие $\nu_1 + \nu_2 + \nu_3 = 0$ будет возникать вращение устройства, но так как в условии задачи не дано расстояние от центра устройства до каждого колеса (только с этим параметром возможно вычисление движение по дуге), мы рассмотрим случай, когда $\nu_1 + \nu_2 + \nu_3 = 0$, в такой ситуации движение устройства прямолинейно и скорость можно вычислить сложением трёх векторов.

$$\nu_x = \nu_1 - \nu_2 \cos(-30^\circ) - \nu_3 \cos(-150^\circ)$$
$$\nu_y = \nu_3 \sin(-150^\circ) - \nu_2 \sin(-30^\circ)$$

Далее вычисление конечных координат происходит по формулам

$$X = \nu_x t$$

$$Y = \nu_y t$$

Задача 6.3.7. Планирование маршрута (5 баллов)

Робот перемещается по полигону, на котором установлены препятствия. Проекции препятствий на поле имеют вид многоугольников, координаты которых задаются во входном файле.

В процессе движения робот может лишь двигаться прямо или поворачиваться на месте вокруг своей оси и не может выходить за пределы полигона.

Определите оптимальный путь перемещения робота из точки старта в точку финиша. Под оптимальным путем подразумевается траектория, состоящая из прямых

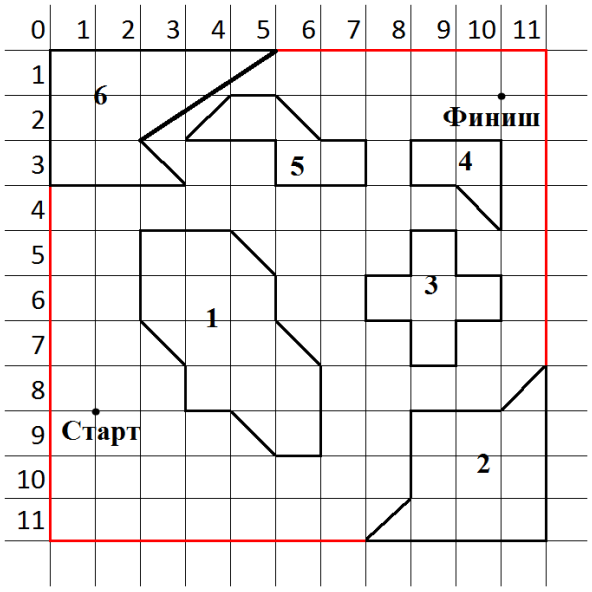


Рис. 6.19: Пример изображения для входных данных

отрезков, где количество отрезков минимально. Концами отрезков являются точки на полигоне, где роботу необходимо выполнить операцию поворота.

Размерами робота можно пренебречь, поэтому допускаются повороты и перемещения вдоль границ полигона и препятствий. На старте и на финише робот может иметь любое направление.

Формат входных данных

Первая строка содержит одно целое число — N — количество многоугольников на полигоне ($0 \leqslant N \leqslant 100$). Вторая строка содержит размер полигона — одно или два целых числа ($10 \leqslant A, B \leqslant 1000$). Третья стока содержит два числа — координаты старта (X_s и Y_s) через пробел. Четвёртая строка содержит два числа — координаты финиша (X_f и Y_f) через пробел.

Далее идёт N блоков. В первой строчке блока находится одно целое число — N_i — количество вершин в данном многоугольнике. После идет N_i строчек, в каждой находится два целых числа — координаты вершины (X_j, Y_j) через пробел. Координаты вершин каждого многоугольника задаются друг за другом по часовой стрелке. Координаты могут быть вещественными числами. Разделитель дробной части - точка.

Формат выходных данных

На первой строчке — количество поворотов K, которые должен совершить робот при перемещении от точки старта к точке финиша. Далее идет K строчек, в каждой из которой указаны координаты точек поворота. Координаты указываются в порядке прохождения точек роботом. Формат вывода координат: X, Y через пробел. В случае если вывести требуется вещественные координаты, то следует указать точность в 1 знак после запятой.

Примеры

Пример №1

Стандартный вход

```
6
11 11
1 8
10 1
11
2 4
4 4
5 5
5 6
6 7
6 9
5 9
4 8
3 8
3 7
2 6
6
11 7
11 11
7 11
8 10
8 8
10 8
12
8 4
9 4
9 5
10 5
10 6
9 6
9 7
8 7
8 6
7 6
7 5
8 5
5
8 2
10 2
10 4
9 3
8 3
8
3 2
4 1
5 1
6 2
```

```
7 2
7 3
6 3
5 2
5
0 0
5 0
2 2
3 3
0 3
```

Стандартный выход

```
3
1 3.5
7.5 3.5
8 1
```

Решение

Для начала мы строим граф по принципу достижимости. Дальше используя BFS мы можем очень просто найти путь с наименьшим числом поворотов, так как повороты происходят только в вершинах нашего графа.

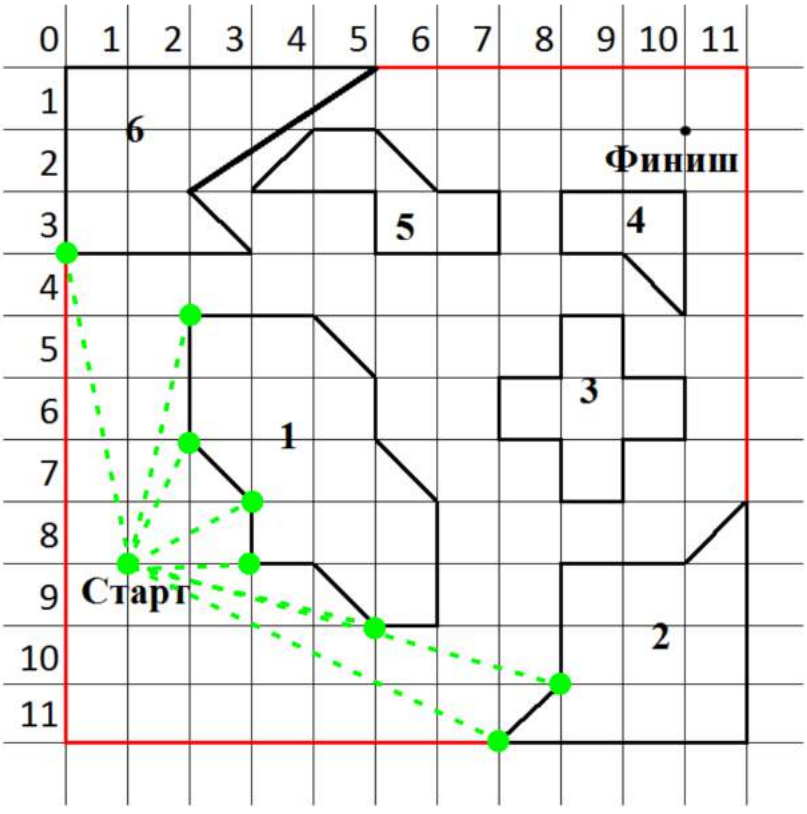


Рис. 6.20: Первая волна постройки графа. Добавляем все вершины в которые можно попасть из точки старта.

Пример программы

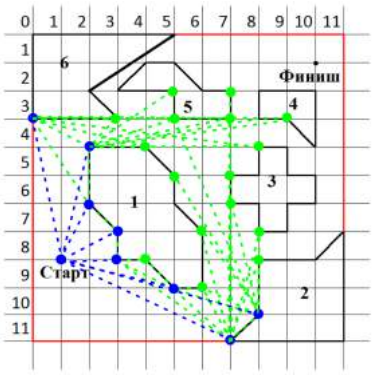
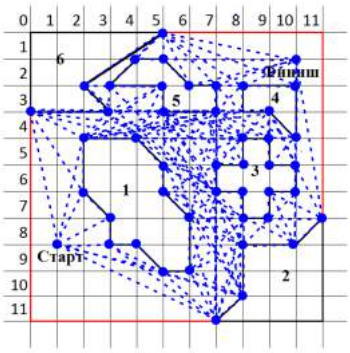
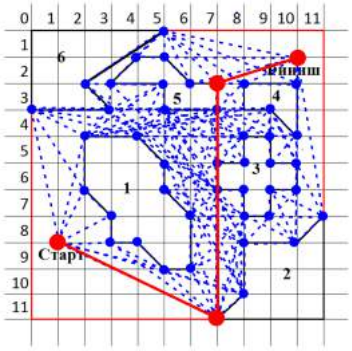


Рис. 6.21: Вторая волна постройки графа. Добавляем все вершины в которые можно попасть из вершин которые уже в графе. Заметим, что так как мы можем двигаться вдоль границ фигур, в нашем графе есть рёбра $(11;7) - (3;7)$ и $(11;7) - (2;7)$.



Конечный вид графа



Путь с наименьшим числом поворотов $(11;7) - (2;7)$

Рис. 6.22:

Ниже представлено решение на языке Java

```
import com.sun.org.apache.xpath.internal.SourceTree;

import java.awt.*;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.io.*;
import java.util.*;
public class Main {
    public static Line2D.Double Povorot(Line2D.Double line, double grad){
        Line2D.Double new_line;
        double x1 = line.x1 , y1 = line.y1 ,x2 = line.x2, y2 = line.y2 , r ;
        grad = (Math.PI*grad)/180;
        r = Math.sqrt( (x2-x1) * (x2-x1) + (y2 - y1) * (y2 - y1));
        x2 = Math.cos(Math.acos((x2 - x1)/r) + grad) * r + x1;
        y2 = Math.sin(Math.asin((y2 - y1)/r) + grad) * r + y1;
        new_line = new Line2D.Double(x1,y1,x2,y2);
        return new_line;
    }
    public static double rastoyanie(Line2D.Double line, Point2D.Double point){
        return (Math.abs( (line.y2-line.y1) * point.x - (line.x2-line.x1) *
                                        point.y + line.x2*line.y1 - line.y2*line.x1)/
                (Math.sqrt( (line.y2-line.y1)*(line.y2-line.y1) + (line.x2 -
                                        line.x1)*(line.x2-line.x1))));
    }
```

```
    public static Point2D.Double Free_point(Point2D.Double[] arr, int koef,
                Line2D.Double dotS, Line2D.Double dotE, Line2D.Double line){
        int k = 0;
        while(Intersection(arr,dotS,dotE,line)) {
            line = Povorot(line, koef);
        }
        Point2D.Double min = null;
        for (int i = 0; i < arr.length; i++)
            if (!(arr[i].x == dotS.x1 && arr[i].y == dotS.y1) &&
                                        !(arr[i].x == dotE.x1 && arr[i].y == dotE.y1))
                min = arr[i];
        for (int i = 1; i < arr.length && min != null; i++)
            if (rastoyanie(line,min) > rastoyanie(line,arr[i]) &&
                                        !(arr[i].x == dotS.x1 && arr[i].y == dotS.y1) &&
                                        !(arr[i].x == dotE.x1 && arr[i].y == dotE.y1))
                min = arr[i];
        return min;
    }
    public static boolean Intersection(Point2D.Double[] arr, Line2D.Double dotS,
                                Line2D.Double dotE, Line2D.Double line) {
        for (int i = 0; i < arr.length; i++) {
            Line2D.Double edge = new Line2D.Double(arr[i].x, arr[i].y,
                    arr[(i + 1) % arr.length].x, arr[(i + 1) % arr.length].y);
            if (edge.intersectsLine(line) && !edge.intersectsLine(dotS)
                                                && !edge.intersectsLine(dotE)) {
                return true;
            }
        }
        return false;
    }
    public static boolean IntersectionAll(Point2D.Double[][] arr,
            Line2D.Double dotS, Line2D.Double dotE, Line2D.Double line){
        boolean flag = false;
        for (int k = 0; k < arr.length && !flag; k++)
            if (Intersection(arr[k], dotS, dotE, line))
                flag = true;
        return flag;
    }

    public static double LenPath(String path){
        if (path.equals("")) return Double.MAX_VALUE;
        String[] s = path.split("\n");
        Point2D.Double[] Points = new Point2D.Double[s.length];
        double result = 0;
        for (int i = 0; i < s.length; i++){
            String[] s1 = s[i].split(" ");
            Points[i] = new Point2D.Double(Double.parseDouble(s1[0]),
                                        Double.parseDouble(s1[1]));
        }
        for (int i = 1; i < Points.length; i++){
            result += Math.sqrt((Points[i-1].x - Points[i].x)*
                                (Points[i-1].x - Points[i].x) +
                    (Points[i-1].y - Points[i].y) * (Points[i-1].y -
                                    Points[i].y));
        }
        return result;
    }
    public static boolean containPart(String s, String p){
        String[] arr = s.split("\n");
        for (int i = 0 ; i < arr.length; i++){
```

```
            if (arr[i].equals(p)) return true;
        }
        return false;
    }
    public static boolean polygon_check(Point2D.Double[] arr,
                        Point2D.Double First, Point2D.Double Second){
        int j1 = 1000,j2 = 1000;
        boolean flag1 = false,flag2 = false;
        for (int i = 0; i < arr.length; i++){
            if (First.x == arr[i].x && First.y == arr[i].y) {
                j1 = i;
                flag1 = true;
            }
            if (Second.x == arr[i].x && Second.y == arr[i].y) {
                j2 = i;
                flag2 = true;
            }
        }
        if (flag1 && flag2){
            return Math.abs(j1-j2)==1 || Math.abs(j1-j2) == arr.length - 1;
        }
        return true;
    }
    public static void Path(Point2D.Double[][] arr,String path_lineS,
                            Point2D.Double lineS,Point2D.Double End) {
        String result = "";
       // System.out.println(lineS.x + " " + lineS.y );
        Line2D.Double dotS = new Line2D.Double(lineS.x,lineS.y,

        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[i].length; j++){
                Point2D.Double lineE = arr[i][j];
                Line2D.Double line = new Line2D.Double(lineS.x, lineS.y,
                                                       lineE.x, lineE.y);
                Line2D.Double dotE = new Line2D.Double(lineE.x,lineE.y,
                                                       lineE.x, lineE.y);
                boolean flag = true;
                for (int k = 0; k < arr.length && flag; k++)
                    if (Intersection(arr[k], dotS, dotE, line))
                        flag = false;
                if ((flag && !containPart(path_lineS,lineS.x + " " + lineS.y))
                                && polygon_check(arr[i],lineS,lineE)) {
                    if (path_point[i][j].split("\n").length >
                            path_lineS.split("\n").length + 1 ||
                            (LenPath(path_point[i][j]) >
                            LenPath(path_lineS + lineS.x + " " +
                                                    lineS.y + '\n')
                            && path_point[i][j].split("\n").length ==
                            path_lineS.split("\n").length + 1  )
                            || path_lineS.equals("") ||
path_point[i][j].equals("")) {
                        path_point[i][j] = path_lineS + lineS.x + " " +
                                                    lineS.y + '\n';
                        Path(arr, path_point[i][j],
                                new Point2D.Double(dotE.x1, dotE.y1), End);
                    }
                }
            }
        }
    }
```

```
145
146     public static Point2D.Double IntersPoint(Line2D.Double line1,
147                                              Line2D.Double line2){
148         double k1,k2,b1,b2,y1,y2,x1,x2;
149         y1 = line1.y1;y2 = line1.y2; x1 = line1.x1; x2 = line1.x2;
150         k1 = (y2 - y1)/(x2 - x1);
151         b1 = y1 - k1 * x1;
152         System.out.println("k1: " + k1 + "   b1:" + b1);
153         y1 = line2.y1;y2 = line2.y2; x1 = line2.x1; x2 = line2.x2;
154         k2 = (y2 - y1)/(x2 - x1);
155         b2 = y1 - k1 * x1;
156         System.out.println("k2: " + k2 + "   b2:" + b2);
157         return new Point2D.Double((b2-b1)/(k1-k2),(b2-b1)/(k1-k2) * k2 + b2);
158     }
159
160     public static Point2D.Double Check(Point2D.Double[][] arr,
161                 Point2D.Double lineS, Point2D.Double lineE, double a, double b){
162         Line2D.Double lineStart = new Line2D.Double(lineS,lineE),
163                                       lineEnd = new Line2D.Double(lineE,lineS);
164         Point2D.Double pp;
165         for (double i = 0; i < b; i+=0.1) {
166             pp = new Point2D.Double(0,i);
167             lineStart = new Line2D.Double(lineS,pp);
168             lineEnd = new Line2D.Double(lineE, pp);
169             if (!IntersectionAll(arr, new Line2D.Double(lineS, lineS),
170                                  new Line2D.Double(pp,pp), lineStart)
171             && !IntersectionAll(arr, new Line2D.Double(lineEnd.getP1(),
172                     lineEnd.getP1()), new Line2D.Double(pp,pp), lineEnd)) {
173                 return pp;
174             }
175         }
176         for (double i = 0; i < a; i+=0.1){
177             pp = new Point2D.Double(i,0);
178             lineStart = new Line2D.Double(lineS,pp);
179             lineEnd = new Line2D.Double(lineE, pp);
180             if (!IntersectionAll(arr, new Line2D.Double(lineS, lineS),
181                                  new Line2D.Double(pp,pp), lineStart)
182                 && !IntersectionAll(arr, new Line2D.Double(lineEnd.getP1(),
183                     lineEnd.getP1()), new Line2D.Double(pp,pp), lineEnd)) {
184                 return pp;
185             }
186         }
187         lineStart = new Line2D.Double(lineS,lineE);lineEnd =
188                                       new Line2D.Double(lineE,lineS);
189         for (double i = 0; i < a; i+=0.1){
190             pp = new Point2D.Double(i,b);
191             lineStart = new Line2D.Double(lineS,pp);
192             lineEnd = new Line2D.Double(lineE, pp);
193             if (!IntersectionAll(arr, new Line2D.Double(lineS, lineS),
194                                  new Line2D.Double(pp,pp), lineStart)
195                     && !IntersectionAll(arr, new Line2D.Double(lineEnd.getP1(),
196                         lineEnd.getP1()), new Line2D.Double(pp,pp), lineEnd)) {
197                 return pp;
198             }
199         }
200         lineStart = new Line2D.Double(lineS,lineE);lineEnd =
201                                       new Line2D.Double(lineE,lineS);
202         for (double i = 0; i < b; i+=0.1){
203             pp = new Point2D.Double(a,i);
204             lineStart = new Line2D.Double(lineS,pp);
```

```
            lineEnd = new Line2D.Double(lineE, pp);
            if (!IntersectionAll(arr, new Line2D.Double(lineS, lineS),
                                    new Line2D.Double(pp,pp), lineStart)
                    && !IntersectionAll(arr, new Line2D.Double(lineEnd.getP1(),
                        lineEnd.getP1()), new Line2D.Double(pp,pp), lineEnd)) {
                return pp;
            }
        }
        return null;
    }
    public static String[][] path_point;
    public static void main(String[] args) throws Exception {
        Locale.setDefault(Locale.US);
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int a = in.nextInt(), b = in.nextInt();
        Point2D.Double Start = new Point2D.Double(in.nextDouble(),

        Point2D.Double End = new Point2D.Double(in.nextDouble(),

        Point2D.Double[][] arr = new Point2D.Double[n+1][];
        path_point = new String[arr.length][];
        for (int i = 0; i < n; i++){
            int m = in.nextInt();
            arr[i] = new Point2D.Double[m];
            path_point[i] = new String[m];
            for (int j = 0 ; j < m; j++) {
                arr[i][j] = new Point2D.Double(in.nextDouble(),

                path_point[i][j] = "";
            }
        }
        arr[n] = new Point2D.Double[1];
        arr[n][0] = End;
        path_point[n] = new String[1];
        path_point[n][0] = "";
        Path(arr,"",Start,End);
        String del = path_point[n][0].split("\n")[0];
        path_point[n][0] = path_point[n][0].replaceFirst(del + "\n","");
        if (path_point[n][0].equals("")){
            System.out.println(0);
        }else {
            Point2D.Double pointcheck = Check(arr,Start,End,a,b);
            if (pointcheck == null || (pointcheck.x < 0 && pointcheck.x > a)
                        || (pointcheck.y < 0 && pointcheck.y > b) ){
                System.out.println(path_point[n][0].split("\n").length);
                System.out.print(path_point[n][0]);
            }else{
                System.out.println(1);
                System.out.println(pointcheck.x + " " + pointcheck.y);
            }
         }
    }
}
```

Задача 6.3.8. Распознавание ARTag маркера (10 баллов)

Для определения своего местоположения квадрокоптер использует камеру, снимающую поверхность, над которой перемещается робот. Изображение с камеры при-

ходит в управляющую программу в видео набора $N \times M$ точек, где каждая точка закодирована в RGB-формате.

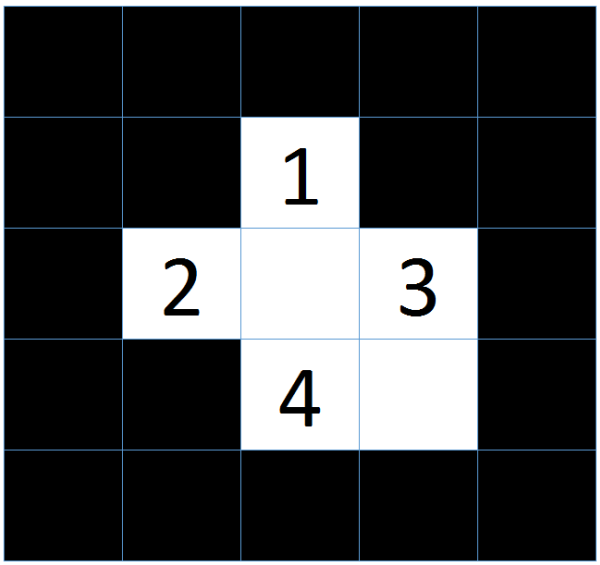


Рис. 6.23: Нумерация битов в маркера

На поверхность нанесены ARTag маркеры `https://inside.mines.edu/~whoff/courses/EENG512/lectures/other/ARTag.pdf`. Элементы маркера, расположенные по его границе - всегда черные. Четыре элемента, находящиеся в углах внутреннего 3×3 квадрата определяют ориентацию маркера таким образом, что только элемент в нижнем правом углу квадрата - белый. Центральный элемент квадрата используется для проверки четности (parity check): если количество единичных бит в двоичной записи закодированного в маркере числа четное, то он черный. Оставшиеся 4 элемента маркера кодируют число по следующему правилу: если элемент черный, то в он обозначает 1, если белый, то 0 при этом самый первый элемент — старший бит закодированного числа. Элементы пронумерованы сверху вниз, слева направо.

Например, на маркере с рис. 6.24 закодировано число 0011_2, что эквивалентно 3_{10}.

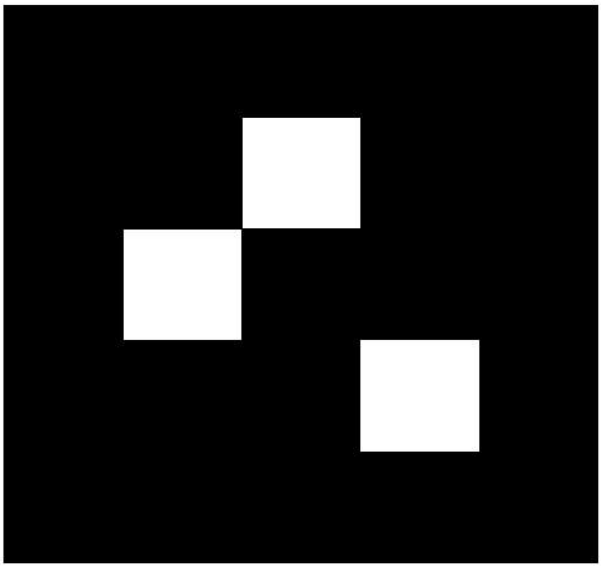


Рис. 6.24: Маркер с закодированным значением - 0011_2

Поскольку квадрокоптер не перемещается постоянно параллельно поверхности, то изображения ARTag маркера, получаемые с камеры, получаются в виде неправильного выпуклого четырехугольника, а непостоянные условия освещенности изменяют фокус и тон изображения.

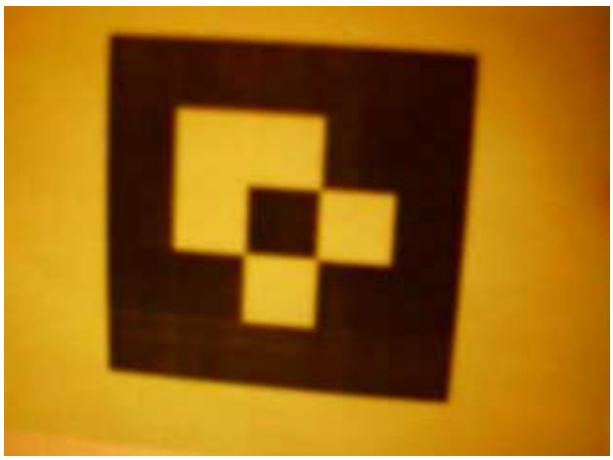


Рис. 6.25: Пример маркера

Поскольку направление запуска квадрокоптера заранее неизвестно, то ориентация маркеров заранее неизвестна, но его изображение таково, что оно по каждой из осей X, Y, Z относительно оптической оси камеры не превышает 25 градусов

Напишите программу для определения закодированного в маркере числа.

Формат входных данных

Первая строка входного файла содержит два целых числа, разделенные пробелом: N и M $(0 \leqslant N, M \leqslant 320)$, определяющие размер кадра.

Далее идет M строк, содержащих N триплетов вида $\#RRGGBB$, где RR - шестнадцатеричное число R составляющей данного элемента матрицы, GG и BB соответственно шестнадцатеричные числа G и B составляющих.

Формат выходных данных

Одно целое число в десятеричной системе счисления — число, закодированное на маркере.

Примеры

Наборы входных данных доступны здесь: `https://drive.google.com/drive/folders/1UOIXYAyd9G_5CES-ePKW_2APlY1LRPzQ`

Решение

Для начала нам нужно перевести изображение в Ч/Б матрицу (бинарный двумерный массив, где 0 – белое, 1 – чёрное), для этого как вариант можно использовать один из самых простых способов, мы записываем в матрицу 1 если сумма всех трёх

компонентов(RGB) меньше какого-то среднего значение, например 120 подходит для всех примеров, следовательно можем сделать вывод, что с большой вероятностью оно подойдёт и для остальных тестов. Далее нам необходимо найти углы маркера, тем самым мы определяем границы маркера, после чего мы делим полученный четырёхугольник на 25 частей, как показано на рисунке

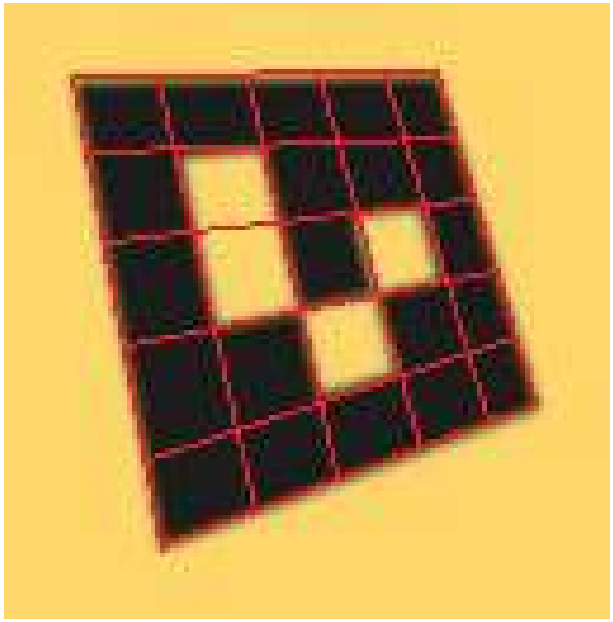


Рис. 6.26: Разметка маркера

Для разметки маркера нам нужнопосторить 12 прямых, каждая прямая задаётся функцией $y = kx + b$. Самыми первыми строятся контурные линии так как нам известны 2 точки на прямой мы можем вычесли k и b по формулам:

$$k = \frac{y_1 - y_2}{x_1 - x_2}$$

$$b = y - kx$$

После вычисления первых четырёх прямых, мы каждую из них делим на 5 равных частей и берём эти точки попарно, что бы найти оставшиеся 8 уравнений. После нахождения коэффициентов всех прямых, нам открывается возможность определять, лежит ли данная точка в данном секторе, если выполняется условие

$$f_l(x) < y < f_r(x)$$

$$f_d(x) < y < f_u(x)$$

Где

- y - координата y точки
- x - координата x точки
- $f_l(x)$ - функция девой границы
- $f_r(x)$ - функция правой границы
- $f_d(x)$ - функция нижний границы
- $f_u(x)$ - функция верхней границы

Далее мы суммируем все пиксили внутри красных четырёхугольниках, и записываем в новую матрицу 25 на 25 1 если чёрных пиксилей больше чем белых и 0 если это не так. После чего нам остаётся определить ориентацию путём нахождения специального бита (в маркере нижний правый угол всегда белого цвета, в то время как все остальные углы чёрного цвета), дальше просто расшифровываем по алгоритму указанному в условие задачи.

Пример программы

Ниже представлено решение на языке C++

```
#include <iostream>
#include <cmath>
#include <stdio.h>

using namespace std;

float PI = 3.14159265;
float PIdiv2 = PI / 2;
float a, blueColor, c;
float xi, yi;

char hexValueColor[8];
int redColor, greenColor, blueColor;
int grayColor = 100;
int N, M;

bool binaryImg[1000][1000];
int borders[1000][1000];
int bordersPoints[100000][2];
int positionCount = 0;
int xMin, yMin, xMax, yMax;
int xMin2, yMin2, xMax2, yMax2;
int xP, yP, xC, yC;
int cornersCoodinates[4][2];
int NumberOfCordes[9][2];
bool matrix[3][3];
int decodeValue = 0;

void RGBRead()
{
    char rhex[] = { hexValueColor[1],hexValueColor[2], };
    char ghex[] = { hexValueColor[3],hexValueColor[4], };
    char bhex[] = { hexValueColor[5],hexValueColor[6], };

    sscanf(rhex, "%x", &redColor);
    sscanf(ghex, "%x", &greenColor);
    sscanf(bhex, "%x", &blueColor);
}

void calculateCoefficients(float x1, float y1, float x2, float y2)
{
    a = y2 - y1;
    blueColor = x1 - x2;
    c = y1*x2 - y2*x1;
}

int solveEquation(float a2, float b2, float c2)
```

```
{
    if (a*b2 - a2*blueColor == 0)
        return 1;
    xi = (c*b2 - c2*blueColor) / (a*b2 - a2*blueColor)*-1;
    yi = (a*c2 - a2*c) / (a*b2 - a2*blueColor)*-1;
    return 0;
}

void findBoarder()
{
    for (int ia = 0; ia < N; ia++)
    {
        for (int ib = 0; ib < M; ib++)
        {
            if (binaryImg[ia][ib] == 1)
            {
                if (ia > yMax)
                {
                    yMax = ia;
                    yMax2 = ib;
                }
                if (ia < yMin)
                {
                    yMin = ia;
                    yMin2 = ib;
                }
                if (ib > xMax)
                {
                    xMax = ib;
                    xMax2 = ia;
                }
                if (ib < xMin)
                {
                    xMin = ib;
                    xMin2 = ia;
                }
                if (borders[ia][ib] != 1)
                {
                    borders[ia][ib] = 1;

                    bordersPoints[positionCount][0] = ib;
                    bordersPoints[positionCount][1] = ia;
                    positionCount++;
                }

                break;
            }
        }
    }

    for (int ia = 0; ia < N; ia++)
    {
        for (int ib = M - 1; ib >= 0; ib--)
        {
            if (binaryImg[ia][ib] == 1)
            {
                if (ia > yMax)
                {
                    yMax = ia;
                    yMax2 = ib;
```

```
                }
                if (ia < yMin)
                {
                    yMin = ia;
                    yMin2 = ib;
                }
                if (ib > xMax)
                {
                    xMax = ib;
                    xMax2 = ia;
                }
                if (ib < xMin)
                {
                    xMin = ib;
                    xMin2 = ia;
                }
                if (borders[ia][ib] != 1)
                {
                    borders[ia][ib] = 1;

                    bordersPoints[positionCount][0] = ib;
                    bordersPoints[positionCount][1] = ia;
                    positionCount++;
                }

                break;
            }
        }
    }

    for (int ib = 0; ib < N; ib++)
    {
        for (int ia = M - 1; ia >= 0; ia--)
        {
            if (binaryImg[ia][ib] == 1)
            {
                if (ia > yMax)
                {
                    yMax = ia;
                    yMax2 = ib;
                }
                if (ia < yMin)
                {
                    yMin = ia;
                    yMin2 = ib;
                }
                if (ib > xMax)
                {
                    xMax = ib;
                    xMax2 = ia;
                }
                if (ib < xMin)
                {
                    xMin = ib;
                    xMin2 = ia;
                }
                if (borders[ia][ib] != 1)
                {
                    borders[ia][ib] = 1;

```

```
                    bordersPoints[positionCount][0] = ib;
                    bordersPoints[positionCount][1] = ia;
                    positionCount++;
                }
                break;
            }
        }
    }

    for (int ib = 0; ib < N; ib++)
    {
        for (int ia = 0; ia < M; ia++)
        {
            if (binaryImg[ia][ib] == 1)
            {
                if (ia > yMax)
                {
                    yMax = ia;
                    yMax2 = ib;
                }
                if (ia < yMin)
                {
                    yMin = ia;
                    yMin2 = ib;
                }
                if (ib > xMax)
                {
                    xMax = ib;
                    xMax2 = ia;
                }
                if (ib < xMin)
                {
                    xMin = ib;
                    xMin2 = ia;
                }
                if (borders[ia][ib] != 1)
                {
                    borders[ia][ib] = 1;

                    bordersPoints[positionCount][0] = ib;
                    bordersPoints[positionCount][1] = ia;
                    positionCount++;
                }

                break;
            }
        }
    }
}

int calculetePoints()
{
    if (binaryImg[yMin][xMin] == 1)
    {
        xP = xMin;
        yP = yMin;
        return 0;
    }
    if (binaryImg[yMin][xMax] == 1)
    {
```

```
228             xP = xMax;
229             yP = yMin;
230             return 0;
231         }
232         if (binaryImg[yMax][xMin] == 1)
233         {
234             xP = xMin;
235             yP = yMax;
236             return 0;
237         }
238         if (binaryImg[yMax][xMax] == 1)
239         {
240             xP = xMax;
241             yP = yMax;
242             return 0;
243         }
244         if (binaryImg[yMax][yMax2] == 1)
245         {
246             xP = yMax2;
247             yP = yMax;
248             return 0;
249         }
250         if (binaryImg[yMin][yMin2] == 1)
251         {
252             xP = yMin2;
253             yP = yMin;
254             return 0;
255         }
256         if (binaryImg[xMax2][xMax] == 1)
257         {
258             xP = xMax;
259             yP = xMax2;
260             return 0;
261         }
262         if (binaryImg[xMin2][xMin] == 1)
263         {
264             xP = xMin;
265             yP = xMin2;
266             return 0;
267         }
268         return 1;
269     }
270
271     int findCorners()
272     {
273         cornersCoodinates[0][0] = xP;
274         cornersCoodinates[0][1] = yP;
275
276         float dist = sqrt(pow(xC - xP, 2) + pow(yC - yP, 2));
277
278         double cosx;
279         double sinx;
280         float a;
281
282         if (dist != 0)
283         {
284             cosx = (yC - xP) / dist;
285             sinx = (xC - yP) / dist;
286         }
287         else
```

```
288             return -2;
289 
290         if (sinx >= 0)
291             a = acos(cosx);
292         else
293             a = 2 * PI - acos(cosx);
294 
295         for (int i = 0; i < 4; i++)
296             {
297             float maxgyp = 0;
298             int mdotx = 0, mdoty = 0;
299             float x1, y1, x2, y2;
300             x1 = xC;
301             y1 = yC;
302             x2 = xC + 10 * cos(a);
303             y2 = yC + 10 * sin(a);
304 
305             calculateCoefficients(x1, y1, x2, y2);
306 
307             for (int ia = 0; ia < positionCount; ia++)
308             {
309                 int xd, yd;
310                 xd = bordersPoints[ia][0];
311                 yd = bordersPoints[ia][1];
312 
313                 float dist = sqrt(pow(xC - xd, 2) + pow(yC - yd, 2));
314 
315                 double cosx;
316                 double sinx;
317                 float ad;
318 
319                 if (dist != 0)
320                 {
321                     cosx = (yd - yC) / dist;
322                     sinx = (xd - xC) / dist;
323                 }
324                 else
325                     return -2;
326 
327                 if (sinx >= 0)
328                     ad = acos(cosx);
329                 else
330                     ad = 2 * PI - acos(cosx);
331 
332                 float dx = ad - a;
333 
334                 if (sin(dx) < 0)
335                 {
336                     solveEquation(0, 1, (yd*-1));
337                     float gyp = sqrt(pow(xi - xd, 2) + pow(yi - yd, 2));
338 
339                     if (gyp > maxgyp)
340                     {
341                         maxgyp = gyp;
342                         mdotx = xd;
343                         mdoty = yd;
344                     }
345                 }
346             }
347 
```

```
348         cornersCoodinates[i][0] = mdotx;
349         cornersCoodinates[i][1] = mdoty;
350 
351         a = a + PIdiv2;
352     }
353 }
354 
355 void GetNumberOfCoords()
356 {
357     int realcenterx, realcentery;
358     float a2, b2, c2;
359 
360     calculateCoefficients(cornersCoodinates[0][0],
361             cornersCoodinates[0][1], cornersCoodinates[2][0],
362             cornersCoodinates[2][1]);
363     a2 = a; b2 = blueColor; c2 = c;
364     calculateCoefficients(cornersCoodinates[1][0],
365             cornersCoodinates[1][1], cornersCoodinates[3][0],
366             cornersCoodinates[3][1]);
367 
368     solveEquation(a2, b2, c2);
369     realcenterx = xi; realcentery = yi;
370 
371     int CucQuad[4][2];
372     for (int i = 0; i < 4; i++)
373     {
374         CucQuad[i][0] = (cornersCoodinates[i][0] - realcenterx) / 2.4 + realcenterx;
375         CucQuad[i][1] = (cornersCoodinates[i][1] - realcentery) / 2.4 + realcentery;
376     }
377 
378     NumberOfCordes[2][0] = CucQuad[0][0];
379     NumberOfCordes[2][1] = CucQuad[0][1];
380     NumberOfCordes[1][0] = (CucQuad[0][0] + CucQuad[1][0]) / 2;
381     NumberOfCordes[1][1] = (CucQuad[0][1] + CucQuad[1][1]) / 2;
382 
383     NumberOfCordes[0][0] = CucQuad[1][0];
384     NumberOfCordes[0][1] = CucQuad[1][1];
385     NumberOfCordes[3][0] = (CucQuad[1][0] + CucQuad[2][0]) / 2;
386     NumberOfCordes[3][1] = (CucQuad[1][1] + CucQuad[2][1]) / 2;
387 
388     NumberOfCordes[6][0] = CucQuad[2][0];
389     NumberOfCordes[6][1] = CucQuad[2][1];
390     NumberOfCordes[7][0] = (CucQuad[2][0] + CucQuad[3][0]) / 2;
391     NumberOfCordes[7][1] = (CucQuad[2][1] + CucQuad[3][1]) / 2;
392 
393     NumberOfCordes[8][0] = CucQuad[3][0];
394     NumberOfCordes[8][1] = CucQuad[3][1];
395     NumberOfCordes[5][0] = (CucQuad[3][0] + CucQuad[0][0]) / 2;
396     NumberOfCordes[5][1] = (CucQuad[3][1] + CucQuad[0][1]) / 2;
397 
398     NumberOfCordes[4][0] = realcenterx;
399     NumberOfCordes[4][1] = realcentery;
400 }
401 
402 
403 void decode()
404 {
405     int icuc = 0;
406 
407     for (int iy = 0; iy<3; iy++)
```

```
408     {
409         for (int ix = 0; ix<3; ix++)
410         {
411             matrix[ix][iy] = binaryImg[NumberOfCordes[icuc][1]][NumberOfCordes[icuc][0]];
412             icuc++;
413         }
414     }
415 
416     if (matrix[2][2] == 0)
417     {
418     bool code[4] = { matrix[1][2],matrix[2][1],matrix[0][1],matrix[1][0], };
419         for (int i = 0; i<4; i++)
420         {
421             decodeValue += pow(2, i)*code[i];
422         }
423     }
424     if (matrix[0][2] == 0)
425     {
426     bool code[4] = { matrix[0][1],matrix[1][2],matrix[1][0],matrix[2][1], };
427         for (int i = 0; i<4; i++)
428         {
429             decodeValue += pow(2, i)*code[i];
430         }
431     }
432     if (matrix[0][0] == 0)
433     {
434     bool code[4] = { matrix[1][0],matrix[0][1],matrix[2][1],matrix[1][2], };
435         for (int i = 0; i<4; i++)
436         {
437             decodeValue += pow(2, i)*code[i];
438         }
439     }
440     if (matrix[2][0] == 0)
441     {
442     bool code[4] = { matrix[2][1],matrix[1][0],matrix[1][2],matrix[0][1], };
443         for (int i = 0; i<4; i++)
444         {
445             decodeValue += pow(2, i)*code[i];
446         }
447     }
448 }
449 
450 
451 int main()
452 {
453     cin >> M >> N;
454 
455     for (int ia = 0; ia < N; ia++)
456     {
457         for (int ib = 0; ib < M; ib++)
458         {
459         cin >> hexValueColor; RGBRead();
460             if (redColor < grayColor && greenColor < grayColor && blueColor < grayColor)
461             {
462             binaryImg[ia][ib] = 1;
463             xMin = ib; xMax = ib; xMin2 = ia; xMax2 = ia;
464             yMin = ia; yMax = ia; yMin2 = ib; yMax2 = ib;
465             }
466         else
467             binaryImg[ia][ib] = 0;
```

```
468             }
469         }
470 
471         findBoarder();
472         if (calculetePoints() == 1) cout << "[ ! ] getPoints() failed\n";
473 
474         xC = (xMax + xMin) / 2;
475         yC = (yMax + yMin) / 2;
476 
477         findCorners();
478         GetNumberOfCoords();
479 
480         decode();
481 
482         cout << decodeValue << endl;
483     }
```