## §3 Финальный этап

Заключительный этап олимпиады состоит из двух частей: индивидуальное решение задач по предметам (математика, информатика) и командное решение инженерной задачи.

## Задачи индивидуального тура

На индивидуальное решение задач дается по 2 часа на один предмет. Для каждого из параллелей (9 класс или 10-11 класс) предлагается свой набор задач по математике, задачи по информатике - общие для всех участников.

Решение каждой задачи по математике дает определенное количество баллов (см. критерии оценки). При этом каждая задача делилась на 3 подзадачи таким образом, что решение каждой подзадачи подводило к решению следующей. За каждую подзадачу можно получить от 0 до указанного количества баллов.

Решение задач по информатике предполагало написание программ. Ограничения по используемым языкам программирования не было. Проверочные тесты для каждой задачи по информатике делились на несколько групп. Прохождение всех тестов в группе группа тестов дает определенное количество баллов за решение задачи.

Участники получают оценку за решение задач в совокупности по всем предметам данного профиля (математика и информатика) — суммарно от 0 до 200 баллов.

## 3.1. Задачи по математике (9 класс)

**Задача 3.1.1 (33 балла)**

*Условие:*

На складе детали упакованы в ящики по 17 или 20 штук.

a. **(3 балла)** Может ли робот-погрузчик загрузить 299 деталей, не вскрывая ящики?
b. **(10 баллов)** Может ли робот-погрузчик загрузить 263 детали, не вскрывая ящики?
c. **(20 баллов)** Какое наибольшее количество деталей робот-погрузчик не сможет загрузить, не вскрывая ящики?

*Решение:*

a. Да, может. 9·20+7·17=299. Сперва заметим, что 6·20-7·17=1 и 15·20=300. В последнем равенстве 6 ящиков по 20 деталей заменим на 7 ящиков по 17 деталей и общее количество деталей уменьшится на 1. (Если пример получен, то необязательно объяснять как именно.)

b. Нет. Пусть робот-погрузчик загрузит x ящиков по 20 деталей и y ящиков по 17. Тогда общее количество загруженных деталей c удовлетворяет равенству

$$20x + 17y = c.(*)$$

Представим c в виде $20 \cdot (6c) + 17 \cdot (-7c)$ и подставим в уравнение

$$20x + 17y = 20 \cdot (6c) + 17 \cdot (-7c);$$
$$20 \cdot (x - 6c) + 17 \cdot (y + 7c) = 0;$$
$$\begin{cases} x - 6c = -17n, \\ y + 7c = 20n, \ n \in; \end{cases}$$

откуда получим все решения (*) в целых числах

$$\begin{cases} x = 6c - 17n, \\ y = 20n - 7c, \ n \in; \end{cases}$$

Но по определению переменные x и y принимают только неотрицательные значения. Поэтому $6c - 17n \geq 0$ и $20n - 7c \geq 0$, что равносильно в свою очередь $\frac{7c}{20} \leq n \leq \frac{6c}{17}$. При c=263 в промежутке

$$92 < \frac{7 \cdot 263}{20} \leq n \leq \frac{6 \cdot 263}{17} < 93$$

целых значений $n$ нет.

c. 303 детали получить нельзя, т.к. в промежутке

$$106 < \frac{7 \cdot 303}{20} \leq n \leq \frac{6 \cdot 303}{17} < 107$$

целых значений n нет. При $c \geq 340$ длина указанного промежутка

$$\frac{6c}{17} - \frac{7c}{20} = \frac{c}{340} \geq 1,$$

поэтому промежуток содержит целое число и найдется решение уравнения (*) в неотрицательных числах. 304=20·5+17·12, меняя ящик с 17-ю деталями на ящик с 20-ю деталями 12 раз получаем распределение по ящикам для 307, 310, 313, 316, 319, 322, 325, 328, 331, 334, 337, 340 деталей. 305=20·11+17·5, из которого легко получаем 308, 311, 314, 317, 320 деталей заменой меньшего ящика большим. 323=17·19, из которого получаем 326, 329, 332, 335, 338 деталей. 306=17·18, откуда получаем кратные трем количество деталей до 20·18=360. Следовательно, любое количество деталей, большее 303, можно загрузить ящиками по 17 и 20 деталей.

*Критерии оценки:*

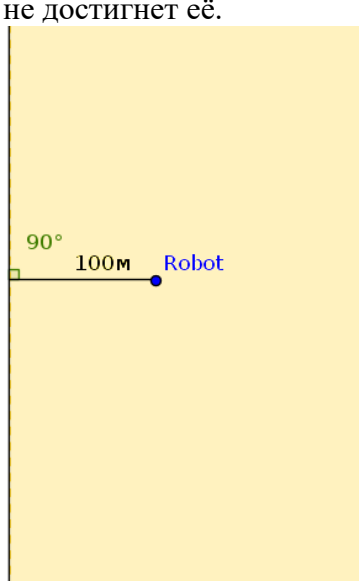
За задачу (b) может быть начислено 10 баллов, если обоснование ответа построено только на базе последней цифры числа 263.

**Задача 3.1.2 (33 балла)**

*Условие:*

Робот находится в поле пшеницы. В памяти сохранилось, что расстояние до границы 100 метров, но направление потеряно. Считать, что поле представляет собой

полуплоскость, т.е. ресурса батареи не хватит выйти из поля в другом направлении, и робот не “увидит” границу поля, пока не достигнет её.

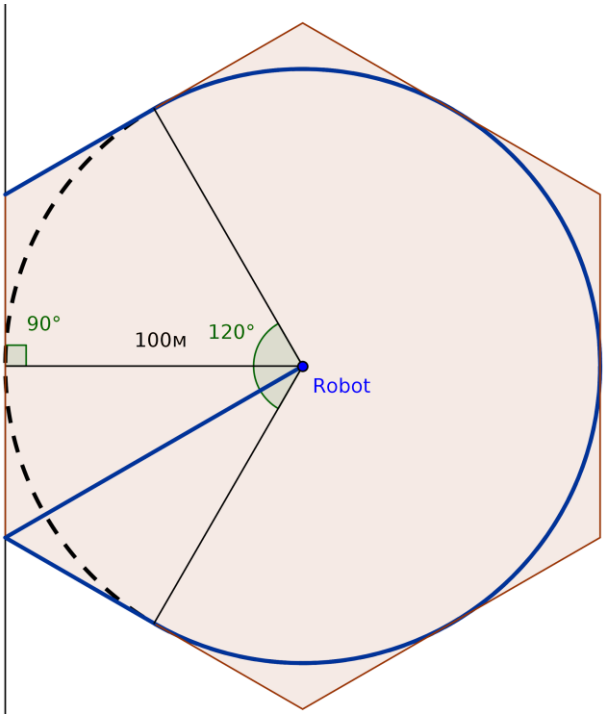



a. **(3 балла)** Существует ли траектория движения робота, которая позволяет выйти из поля, пройдя не более 800 метров;
b. **(13 баллов)** Существует ли траектория движения робота, которая позволяет выйти из поля, пройдя не более 700 метров;
c. **(17 баллов)** Существует ли траектория движения робота, которая позволяет выйти из поля, пройдя не более 650 метров.

*Решение:*

Граница поля является касательной к окружности с центром в точке нахождения робота и радиусом 100 метров. Поэтому достаточно придумать траекторию, которая пересекает все касательные к этой окружности. Достаточно двигаться по периметру многоугольника, описанного вокруг окружности. Меняя количество сторон, можем оптимизировать траекторию.

Во всех пунктах ответ утвердительный. Причем решение пункта в) подходит и для первых двух. Поэтому приведем путь короче 650 метров, которая обязательно выведет робота к границе поля.



Длина синей траектории равна $\frac{2}{\sqrt{3}} \cdot 100 + \frac{1}{\sqrt{3}} \cdot 100 + \frac{2}{3} \cdot 2\pi \cdot 100 + \frac{1}{\sqrt{3}} \cdot 100,$ что меньше 650.

*Критерии оценки:*

За задачу (а) может быть начислен 1 балл, если приведен пример траектории без обоснования.

**Задача 3.1.3 (34 балла)**

*Условие:*

a. **(6 баллов)** Какое преобразование будет результатом последовательного применения трех осевых симметрий относительно биссектрис треугольника ABC?
b. **(12 баллов)** Через центр O окружности проведены три прямые. При помощи циркуля и линейки построить описанный около окружности треугольник, вершины которого лежат на этих прямых.
c. **(16 баллов)** Через центр O окружности проведены 2017 прямых. Как построить описанный около окружности 2017-угольник, вершины которого лежат на этих прямых?

*Решение:*

a. Пусть O - центр вписанной окружности треугольника ABC, OD - высота, опущенная на AC. Рассмотрим композицию $F=S_{CO} \circ S_{BO} \circ S_{AO}$ трех осевых симметрий относительно биссектрис углов A, B, C в указанном порядке. Это преобразование является движением, меняет ориентацию треугольника ABC, оставляет на месте точки O и D, переводит в себя вписанную окружность и прямую AC. Нетрудно проверить, что F является осевой симметрией относительно прямой OD. Действительно, движение задается тремя точками, и если две из них переходят в себя (O и D), то третья перейдет в себя или в симметричную точку (относительно OD), т.к. расстояния сохраняются. В первом случае сохраняется ориентация (тождественное преобразование), во втором меняется (осевая симметрия).
b. Прямые AO, BO и CO нам известны, необходимо выяснить местоположение самих точек A, B и C. Применяя преобразование F к некоторой точке X получим точку Y, серединный перпендикуляр отрезка XY есть прямая OD в обозначениях предыдущего пункта. Касательная к окружности, перпендикулярная OD, пересекает прямые AO и CO в точках A и C соответственно. Вершина B восстанавливается элементарно. Задача имеет два решения, так как существуют две касательные к окружности, перпендикулярные OD.
c. Композиция нечетного количества осевых симметрий относительно прямых с общей точкой является осевой симметрией относительно прямой, проходящей через ту же точку. Доказательство по индукции.
   Обозначим $A_1A_2...A_{2017}$ искомый 2017-угольник. Композиция 2017 осевых симметрий относительно прямых $A_1O$, $A_2O$, ..., $A_{2017}O$ переведет прямую $A_1A_{2017}$ в себя, следовательно, ось симметрии соответствующей этой композиции перпендикулярна прямой $A_1A_{2017}$, её можно восстановить по образу и прообразу некоторой точки. Берем произвольную точку $X_0$, применяем симметрии
$$S_{A_kO}(X_{k-1} = X_k,\ 1 \le k \le 2017,$$

строим серединный перпендикуляр к $X_0X_{2017}$, в точке пересечения этой прямой с окружностью строим касательную, она и есть прямая $A_1A_{2017}$. Остальные стороны 2017-угольника получаем последовательно отражая симметрично предыдущую сторону относительно биссектрисы угла при этой стороне. Задача имеет два решения.

## 3.2 Задачи по математике (10-11 класс)

**Задача 3.2.1 (16 баллов)**

*Условие:*

Саша и Рустам захотели попить кофе. Саша сделал полный стакан кофе (200 мл), а Рустам налил половину стакана молока (100 мл), и тут выяснилось, что в аппарате закончился кофе. Тогда Саша решил поделиться с другом. Он перелил кофе в стакан Рустама из своего, пока тот не заполнился, при этом тщательно перемешивал. Потом это действие повторил в отношении к своему стакану. Эту процедуру Саша повторил несколько раз. В итоге у них в стаканах оказалось столько же жидкости, что в начале, но концентрация кофе отличалась менее чем на 1%. Какое наименьшее количество переливаний сделал Саша?

*Решение:*

Пусть в некоторый момент времени доля молока в стакане у Саши была равна $\alpha_i$ (при этом у него в стакане 200 мл), а у Рустама доля молока равна $\beta_i$ (в стакане 100 мл). Тогда после одной операции переливания в стакане Рустама окажется

$100 \cdot (1 - \beta_i) + 100 \cdot (1 - \alpha_i)$ мл кофе и $100 \cdot (\alpha_i + \beta_i)$ мл молока.

Доли молока и кофе в первом стакане не изменилась (то есть $\alpha_{i+1} = \alpha_i$), изменился только общий объем. А вот во втором стакане доля молока теперь равна

$$\beta_{i+1} = \frac{100 \cdot (\alpha_i + \beta i)}{200} = \frac{\alpha_i + \beta_i}{2}.$$

После обратного переливания в стакане Саши окажется

$$100 \cdot (1 - \alpha_{i+1}) + 100 \cdot (1 - \beta_{i+1}) = 100 \cdot \left(2 - \frac{3\alpha_i}{2} - \frac{\beta_i}{2}\right)$$ мл кофе

и

$$100 \cdot \alpha_{i+1} + 100 \cdot \beta_{i+1} = 100 \cdot \left(\frac{3\alpha_i}{2} + \frac{\beta_i}{2}\right)$$ мл молока.

Следовательно, доля молока равна

$$\alpha_{i+2} = \frac{100 \cdot \left(\frac{3\alpha_i}{2} + \frac{\beta_i}{2}\right)}{200} = \frac{3\alpha_i + \beta_i}{4},$$

а в стакане у Рустама не изменилась

$$\beta_{i+2} = \beta_{i+1} = \frac{\alpha_i + \beta_i}{2}.$$

Теперь посмотрим как изменялся модуль разности концентрации молока: изначально он был равен

$$\gamma_i = |\alpha_i - \beta_i|,$$

а после двух переливаний стал равен

$$\gamma_{i+2} = |\alpha_{i+2} - \beta_{i+2}| = \left|\frac{3\alpha_i + \beta_i}{4} - \frac{\alpha_i + \beta_i}{2}\right| = \frac{|\alpha_i - \beta_i|}{4} = \frac{\gamma_i}{4}.$$

В итоге мы получили, что после двух переливании модуль разности концентрации уменьшается ровно в 4 раза. Изначально он равен $\gamma_0 = 1$, а после k пар переливаний впервые стал менее чем $\gamma_{2k} < \frac{1}{100}$. Следовательно, нужно найти такое наименьшее k, что

$\frac{1}{4^k} < \frac{1}{100}$ (если разность концентрации кофе меньше 1, то и разность концентрации молока тоже меньше 1 и наоборот). Получаем, что k = 4. Поэтому всего переливаний было 8.

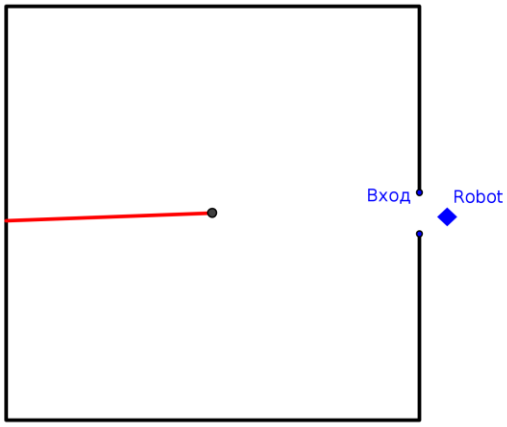
*Критерии оценки:*

- За полный и правильный расчет всех значений концентрации - 16 баллов;
- Если имелась вычислительная ошибка при расчет концентрации - не более 8 баллов;
- Если в какой-то момент времени происходило округление значений - не 14 баллов.

**Задача 3.2.2 (34 балла)**

*Условие:*

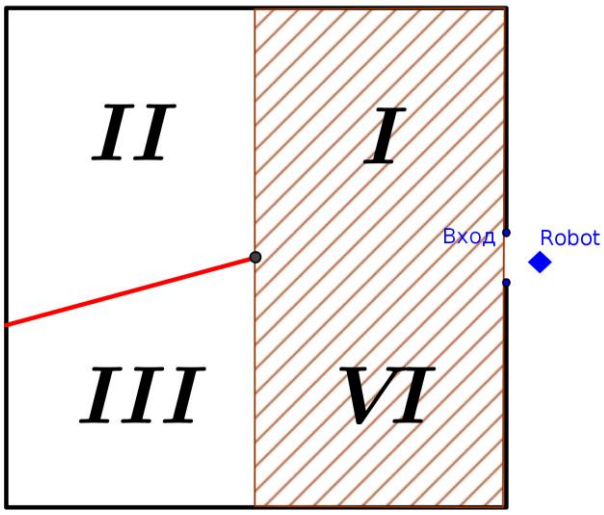
В центре квадратной комнаты со стороной 20 метров находится лазерный радар, который вращается со скоростью один оборот в минуту. Робот должен незаметно пробраться до центра, не попав под луч радара.




a. **(14 баллов)** Докажите, что робот не сможет достичь цели, если его скорость меньше 8 м/мин.
b. **(20 баллов)** Сможет ли робот достичь цели, если его скорость меньше 10 м/мин.

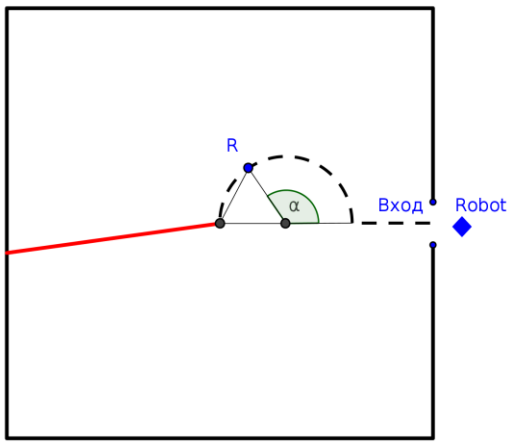
*Решение:*

a. Не умаляя общности будем считать, что радар вращается против часовой стрелки. Если радар в начальный момент времени (момент начала движения робота) в VI четверти, то он за половину минуты дойдет до конца I четверти и пересечет робота.




А если в I, II или III четверти, то за 1.25 минут он точно покроет заштрихованную область (непрерывно, начиная с начала IV и заканчивая концом I четверти). Так как скорость робота меньше 8 м/мин, то за 1.25 минут он удалится менее чем на 1.25·8 = 10 м от начальной точки, а значит будет в заштрихованной области. Следовательно, робот попадет под луч радара.

b. Пусть робот выехал ровно в тот момент, когда радар начинал движение в $I$ четверти. Также будем считать, что робот двигался по траектории обозначенной пунктиром. Причем, весь путь он двигался с постоянной скоростью v = 9.5 м/мин, t = 0.75 мин двигался прямолинейно, а остаток пути по полуокружности. Докажем, что робот не попадет под радар. На прямолинейном участке пути радар не найдет робота, так как за 0.75 мин он не дойдет до конца IV четверти. Рассмотрим теперь произвольную точку R на полуокружности. Пусть центральный угол равен $\alpha$, тогда точка R видна под углом $\frac{\alpha}{2}$ с центра поля (отсчет начинается с начала I четверти).




Радар не засечет робота в точке R если

$$t + \frac{(10 - vt)\pi}{2v} \cdot \frac{\alpha}{\pi} < 1 + \frac{\alpha}{4\pi}$$

Подставим значения

$$0.75 + \frac{23}{152} \cdot \alpha < 1 + 0.25 \cdot \frac{\alpha}{\pi} \iff \left(\frac{23\pi}{152} - \frac{1}{4}\right) \cdot \frac{\alpha}{\pi} < 0.25$$

Последнее неравенство очевидно верно для $\alpha \in [0, \pi]$. Следовательно, на полуокружности радар не засечет робота.

*Критерии оценки:*

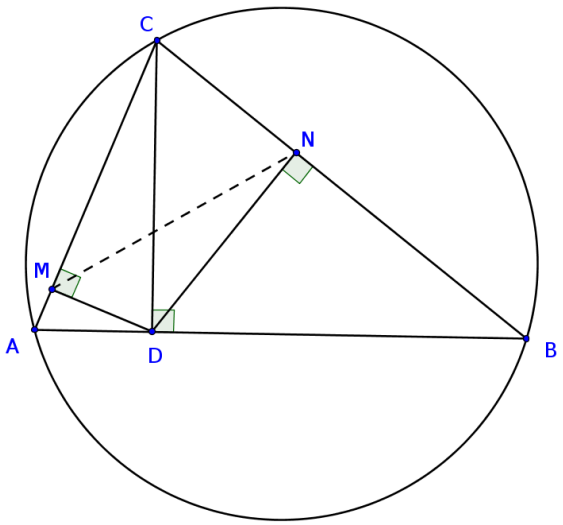
- Если в задаче (a) есть только утверждение о том, в какой момент времени должен поехать робот - 2 балла;
- Если в задаче (a) рассмотрена только окрестность расположения робота через некоторое время - 5 баллов;
- Если в задаче (b) указана только траектория движения робота без доказательства корректности - не более 10 баллов.

**Задача 3.2.3 (50 баллов)**

*Условие:*

a. **(14 баллов)** Точки A(a) и B(b) лежат на единичной окружности $z \cdot \overline{z} = 1$ с центром в начале координат комплексной плоскости. Докажите, что координата ортогональной проекции точки M(m) на прямую AB задается уравнением

$$z = \frac{1}{2}(a + b + m - ab\overline{m})$$




b. **(16 баллов)** Пусть дан треугольник ABC (см. рисунок), вершины которого соответствуют комплексным числам a, b и c, причём описанная около него окружность имеет уравнение $z \cdot \overline{z} = 1$. Из основания высоты CD треугольника опущены перпендикуляры DM и DN на две стороны. Докажите, что

$$m - n = \frac{(a-b)(a-c)(b-c)}{4ab},$$

где m и n - комплексные координаты точек M и N.

c. **(20 баллов)** Докажите, что длина MN не зависит от выбора высоты треугольника.

*Решение:*

a. Найдём координату ортогональной проекции точки M(m) на прямую, заданную точками A(a) и B(b). Если Z(z) -- искомая точка, то имеем

$$\begin{cases} z(\overline{a} - \overline{b}) + a(\overline{b} - \overline{z}) + b(\overline{z} - \overline{a}) = 0 \ (1) \\ (a-b)(\overline{m} - \overline{z}) + (\overline{a} - \overline{b})(m - z) = 0 \ (2) \end{cases}$$

где (1) - условие коллинеарности трех точек A, B, Z; (2) - условие ортогональности $AB \perp MZ$.

Откуда выражаем z и получаем

$$z = \frac{a(\overline{m} - \overline{b}) - b(\overline{m} - \overline{a})}{2(\overline{a} - \overline{b})} + \frac{m}{2}$$

В случае, когда точки A(a) и B(b) принадлежат единичной окружности с центром в начале координат $a\overline{a} = b\overline{b} = 1$. Поэтому

$$z = \frac{a(\overline{m} - \overline{b}) - b(\overline{m} - \overline{a})}{2(\overline{a} - \overline{b})} + \frac{m}{2} = \frac{\overline{a}b - a\overline{b}}{2(\overline{a} - \overline{b})} + \overline{m} \cdot \frac{a - b}{2(\overline{a} - \overline{b})} + \frac{m}{2} =$$
$$= \frac{a + b}{2} - \overline{m} \cdot \frac{ab}{2} + \frac{m}{2} = \frac{1}{2}(a + b + m - ab\overline{m})$$

b. Так как $a\overline{a} = b\overline{b} = c\overline{c} = 1$, то воспользуемся три раза предыдущим пунктом для точки D с отрезками AC, BC и для точки C с отрезком AB

$$\begin{cases} m = \frac{1}{2}(a + c + d - ac\overline{d}) \\ n = \frac{1}{2}(b + c + d - bc\overline{d}) \\ d = \frac{1}{2}(a + b + c - ab\overline{c}) \end{cases} \Longrightarrow$$

$$\Longrightarrow m - n = \frac{1}{2}(a + c + d - ac\overline{d}) - \frac{1}{2}(b + c + d - bc\overline{d}) = \frac{1}{2}(a - b)(1 - c\overline{d}) =$$
$$= \frac{(a - b)}{4}(2 - c(\overline{a} + \overline{b} + \overline{c}) + \overline{a}\overline{b}cc) = \frac{(a - b)}{4}\frac{(ab - cab(\overline{a} + \overline{b}) + cc)}{ab} =$$
$$\frac{(a - b)(a - c)(b - c)}{4ab}$$

c. Так как $|a| = |b| = 1$, то

$$|m - n| = \frac{|a - b| \cdot |b - c| \cdot |c - a|}{4|a| \cdot |b|} = \frac{|a - b| \cdot |b - c| \cdot |c - a|}{4}$$

Это выражение симметрично относительно a, b и c, поэтому длина MN не зависит от выбора высоты треугольника.

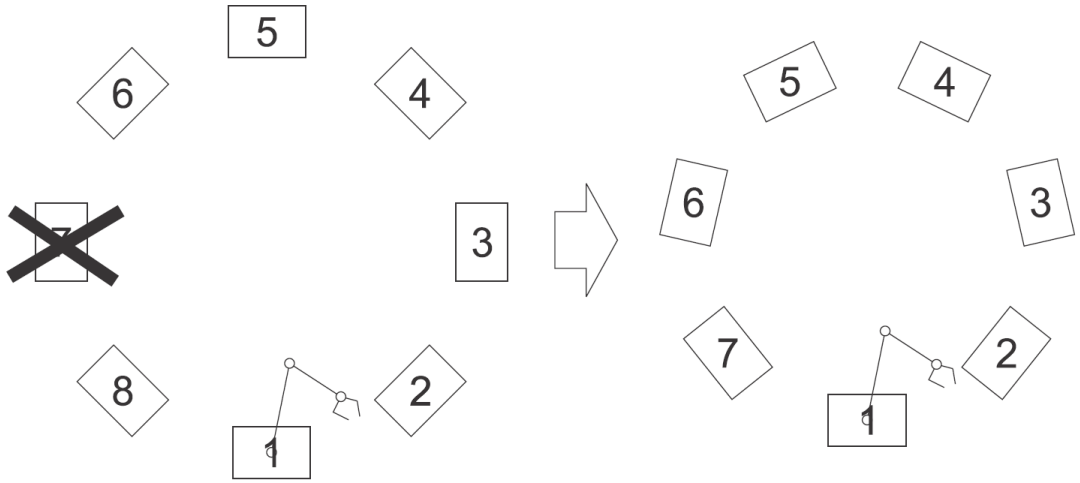
## 3.3 Задачи по информатике

**Задача 3.3.1 (30 баллов)**

*Условие:*

В цехе по упаковке конечной продукции на заводе по производству мебели робот-манипулятор подготавливает товар к отгрузке. Контейнеры с элементами, из которых упаковывается та или иная единица мебели, расположены так, что робот вместе с ними образует круг. Каждая позиция, где может быть расположен контейнер, пронумерована против часовой стрелки. Манипулятор установлен в позиции 1, справа от него находится контейнер с позицией 2, слева — контейнер с позицией n.

В каждом i-ом контейнере находится $c_i$ определенного типа элементов. Во время операций по отгрузке робот может доставать из контейнеров по одному элементу. Длины звеньев манипулятора хватает только, чтобы достичь k-го контейнера по правую или по левую сторону от робота. Если во время операций по отгрузке какой-то контейнер становится пустым, он отправляется в производственных цех, а круг контейнеров сужается.




Операция извлечения элемента и перемещения его в зону упаковки манипулятором занимает 1 минуту.

В какой-то момент времени в цех по упаковке пришел запрос из производственного цеха освободить контейнер с определенным типом элементов. Необходимо определить, какое минимальное количество времени необходимо манипулятору, чтобы освободить запрошенный контейнер, находящийся на текущий момент в позиции под номером m.

**Формат входных данных:**

- Первый набор тестов:
  В первой строке входных файлов содержится три разделенных пробелом целых числа n, k, m $(2 \le n \le 25,\ 1 \le kn,\ 2 \le m \le n)$.
  Во второй строке содержится n разделенных пробелом чисел, где на i-й позиции стоит $h_i$ $(1 \le h_i \le 10\,000)$ — количество элементов в i-ом контейнере.
- Второй набор тестов:
  В первой строке входных файлов содержится три разделенных пробелом целых числа n, k, m $(2 \le n \le 500,\ 1 \le kn,\ 2 \le m \le n)$.
  Во второй строке содержится n разделенных пробелом чисел, где на i-й позиции стоит $h_i$ $(1 \le h_i \le 10\,000)$ — количество элементов в i-ом контейнере.
- Третий набор тестов:
  В первой строке входных файлов содержится три разделенных пробелом целых числа n, k, m $(2 \le n \le 10\,000,\ 1 \le kn,\ 2 \le m \le n)$.
  Во второй строке содержится n разделенных пробелом чисел, где на i-й позиции стоит $h_i$ $(1 \le h_i \le 10\,000)$ — количество элементов в i-ом контейнере.

**Формат выходных данных:**

Выведите одно число — минимальное количество времени, необходимое для освобождения контейнера под номером m.

**Пример №1:**

***stdin:***
6 2 4
10 3 2 5 4 4
***stdout:***
7

**Пример №2:**

***stdin:***

5 1 5
1 4 8 2 6
***stdout:***
6

**Примечание:**

В первом примере манипулятор не может сразу достигнуть до контейнера, поэтому для начала ему нужно потратить 2 минуты, чтобы освободить контейнер под номером 3. После этого манипулятор уже может выгрузить нужный контейнер, потратив 5 минут.

Во втором примере манипулятор сразу может за 6 минут выгрузить контейнер.

*Критерий оценки:*

За решение задачи начислялось:

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

Для проверки результата используется следующий код на языке Python:

```
def solve(dataset):
    dataset = dataset.splitlines()
    n, k, m = map(int, dataset[0].split())
    m = m - 1
    health = list(map(int, dataset[1].split()))

    heap = []
    r_ans = 0
    last = k
    for i in range(1, last + 1):
        heappush(heap, health[i])
    while last < m:
        r_ans += heappop(heap)
        last = last + 1
        heappush(heap, health[last])

    heap = []
    l_ans = 0
    last = n - k
    for i in range(last, n):
        heappush(heap, health[i])
    while last > m:
        l_ans += heappop(heap)
        last = last - 1
        heappush(heap, health[last])

    return str(min(l_ans, r_ans) + health[m])
```

*Решение:*

Немножко изменим нашу задачу, пусть контейнеры стоят в ряд, а не создают круг. Манипулятор может достать контейнеры от 1 до k. Какой контейнер выгоднее опустошить, чтобы манипулятор доставал до контейнера k+1? Очевидно, что контейнер с наименьшим количеством элементов в нем, при этом другие контейнеры мы не трогаем.

А если хотим достать до контейнера k+2? Тогда требуется выбрать контейнер с минимальным количеством элементов от 1 до k и еще один минимум от 1 до k+1, но без учета контейнера, который мы уже обработали.

К чему свелась наша задача? Каждый раз нам требуется находить минимум среди k элементов, удалять его, и повторять это действие до тех пор, пока не достанем до нужного контейнера. Структура данных двоичная куча позволяет добавлять элементы за O(log(k)) и убирать минимум за такую же асимптотику. Итоговое время работы: O(n·log(k)).

Так как наши контейнеры образуют круг, требуется еще запустить алгоритм, но уже брать контейнеры с другой стороны.

Пример программы, реализующей данный алгоритм на языке C++:

```
#include <iostream>
#include <vector>
#include <set>
#include <cmath>
#include <cstdio>

using namespace std;

int main() {
    int n, k, m;
    cin >> n >> k >> m;
    vector <int> health(n + 1);
    for (int i = 1; i <= n; i++) {
        scanf("%d ", &health[i]);
    }

    multiset <int> heap;
    int r_ans = 0;
    int last = k + 1;
    for (int i = 2; i <= last; i++) {
        heap.insert(health[i]);
    }
    while (last < m) {
        r_ans += *(heap.begin());
        heap.erase(heap.begin());
        heap.insert(health[++last]);
    }

    heap.clear();
    int l_ans = 0;
    last = n - k + 1;
    for (int i = n; i >= last; i--) {
        heap.insert(health[i]);
    }
    while (last > m) {
        l_ans += *(heap.begin());
        heap.erase(heap.begin());
        heap.insert(health[--last]);
    }

    cout << min(l_ans, r_ans) + health[m] << endl;
}
```

**Задача 3.3.2 (30 баллов)**

*Условие:*

Передача информации от Центра Планирования Миссии (ЦПМ) до робота на Марсе происходит пакетами. Пакеты передаются в виде массива, каждый пакет представляет из себя целое положительное число. Пакеты доходят с разной скоростью, и из-за этого существует одна проблема — они могут поменяться местами. Но мы живём в будущем, и не всё уж так плохо: гарантируется, что пакет мог поменяться только с одним из соседних пакетов (после чего эти 2 пакета не могли дальше поменяться местами с новыми соседями).

Главе ЦПМ надо приготовить отчёт, поэтому ему надо узнать, какой может быть максимальное значение суммы, вычисленной по следующей формуле: $a_1$ xor $a_2$ + $a_3$ xor $a_4$ + ... + $a_{n-1}$ xor $a_n$, где xor - побитовое исключающее ИЛИ. Именно вам предстоит вычислить это значение.

**Формат входных данных:**

- Первый набор тестов
  В первой строке дано целое положительное число n $(2 \leqslant n \leqslant 20)$ — количество элементов массива. Гарантируется, что оно чётное.
  Следующая строка содержит n чисел — изначально заданный массив.
  Все числа положительные и не превосходят $10^5$.
- Второй набор тестов
  В первой строке дано целое положительное число n $(2 \leqslant n \leqslant 100\,000)$ — количество элементов массива. Гарантируется, что оно чётное.
  Следующая строка содержит n чисел — изначально заданный массив.
  Все числа положительные и не превосходят $10^5$.

**Формат выходных данных:**

Выведите одно число — максимальное значение суммы, вычисленный вышеописанным способом.

**Пример №1:**

***stdin:***
4
1 1 2 6
***stdout:***
10

**Пример №2:**

***stdin:***
6
5 7 6 1 8 4
***stdout:***
23

**Примечание:**

Массив в первом пример, при котором достигается максимальная сумма: 1 2 1 6.
Во втором примере: 5 6 7 8 1 4.

*Критерий оценки:*

За решение задачи начислялось:

- **10 баллов**, если пройдена только первая группа тестов;

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

Для проверки результата используется следующий код на языке Python:

```
def solve(dataset):
    dataset = dataset.splitlines()
    n = int(dataset[0])
    a = dataset[1].split()

    for i in range(0, n):
        a[i] = int(a[i])

    a = [a[0]] + a + [a[n - 1]]
    n = n + 2

    dp = [[0 for x in range(2)] for y in range(n)]

    for i in range(3, n, 2):
        dp[i][0] = max(dp[i - 2][0] + (a[i - 2] ^ a[i - 1]),
                       dp[i - 2][1] + (a[i - 3] ^ a[i - 1]))
        dp[i][1] = max(dp[i - 2][0] + (a[i - 2] ^ a[i]),
                       dp[i - 2][1] + (a[i - 3] ^ a[i]))

    return str(max(dp[n - 1][0], dp[n - 1][1]))
```

*Решение:*

Заведем матрицы $dp_0[n]$, $dp_1[n]$ – максимальный ответ, если разрешено использовать только первые i элементов, при этом $dp_0[i]$ – мы не меняем местами (i-1)-й и i-й элементы, $dp_1[i]$ – меняем. Тогда как получить ответ для j, если мы знаем ответ для префикса?

$$dp_0[j] = max(dp_0[j-2]+(a[j-2]\oplus a[j-1]), dp_1[j-2]+(a[j-3]\oplus a[j-1]))$$
$$dp_1[j] = max(dp_0[j-2] + (a[j-2] \oplus a[j]), dp_1[j-2] + (a[j-3] \oplus a[i]))$$

Пример программы, реализующей данный алгоритм на языке C++:

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n;
    scanf("%ld", &n);

    vector<long long> a(n + 2);

    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
    }

    n += 2;
    a[0] = a[1];
    a[n - 1] = a[n - 2];

    vector< vector<long long> > dp(n, vector<long long> (2, 0));

    for (int i = 3; i < n; i += 2) {
        dp[i][0] = max(dp[i - 2][0] + (a[i - 2] ^ a[i - 1]),
                       dp[i - 2][1] + (a[i - 3] ^ a[i - 1]));
        dp[i][1] = max(dp[i - 2][0] + (a[i - 2] ^ a[i]),
```

```
                    dp[i - 2][1] + (a[i - 3] ^ a[i]));
    }

    printf("%lld", max(dp[n - 1][0], dp[n - 1][1]));

    return 0;
}
```

**Задача 3.3.3 (40 баллов)**

*Условие:*

В одном из государств было решено создать сеть из n городов, между которыми бы курсировали беспилотные автомобили, но остается проблема возникновения внештатных ситуаций. Для решения этого вопроса было решено поставить маяки, которые бы управляли автомобилями, в случаях возникновения опасности. В каждом из n городов хотят поставить по одному маяку так, чтобы если какой-то один маяк выйдет из строя, то оставшиеся города тоже представляли из себя единую сеть, что из всех оставшихся существовал путь между любыми двумя городами.

Так как государство не хочет сильно тратиться, то решили произвести одинаковые маяки с наименьшем радиусом действия, но выполняющие все условия безопасности, которые были описаны ранее.

В нашем случае представим, что города — точки на плоскости, а дороги, как кратчайшее расстояние между точками. Заметим, что из города a в город b можно попасть и через другие города. В итоге мы хотим выбрать минимальный радиус маяка, чтобы получить такую сеть, что из любого города можно добраться до любого другого и, если какой-то маяк выйдет из строя, то оставшиеся города тоже представляли из себя единую сеть.

**Формат входных данных:**

- Первый набор тестов
  Первая строка содержит одно целое число n $(0 \leqslant n \leqslant 100)$ — количество городов.
  Следующие n строк содержат по два числа — координаты городов соответственно.
  Все координаты точек неотрицательные и не превосходят $10^9$.
- Второй набор тестов
  Первая строка содержит одно целое число n $(0 \leqslant n \leqslant 300)$ — количество городов.
  Следующие n строк содержат по два числа — координаты городов соответственно.
  Все координаты точек неотрицательные и не превосходят $10^9$.

**Формат выходных данных:**

Выведите одно число — минимальный радиус действия для всех маяков, удовлетворяющий всем вышеописанным условиям. Число выведите с точностью не менее $10^{-6}$.

**Пример №1:**

***stdin:***

4
3 0
0 4
3 5

6 4
***stdout:***
2.500000

**Пример №2:**

***stdin:***
6
2 1
3 5
8 2
8 4
12 1
12 5
***stdout:***
3.041381

*Критерий оценки:*
За решение задачи начислялось:

- **15 баллов**, если пройдена только первая группа тестов;
- **40 баллов**, если пройдены все тесты.

Для проверки результата используется следующий код на языке Python:

```
def dfs(v, p, edges, used, tin, up):
    global t
    used[v] = 1
    up[v] = t
    tin[v] = t
    t = t + 1
    flag = 0
    was = 0
    child = 0
    for i in range(len(edges[v])):
        to = i
        if to == p or edges[v][i] == 0:
            continue
        if used[to] == 0:
            child = child + 1
            was = max(was, dfs(to, v, edges, used, tin, up))
            up[v] = min(up[v], up[to])
            if up[i] >= tin[v]:
                flag = 1
        else:
            up[v] = min(up[v], tin[to])
    if p == -1:
        flag = 0
    if flag == 1 or (p == -1 and child > 1):
        return 1
    else:
        return was


def solve(dataset):
    global t
    dataset = dataset.splitlines()
    n = int(dataset[0])
    a = [list(map(int, dataset[i + 1].split())) for i in range(n)]
```

```
    t = 0
    l = -1
    r = 10 ** 18
    while r - l > 1:
        edges = [[0 for i in range(n)] for j in range(n)]
        used = [0] * n
        up = [0] * n
        tin = [0] * n
        t = 0
        m = (l + r) // 2
        for i in range(n):
            for j in range(n):
                dist = (a[i][0] - a[j][0]) ** 2 +
                       (a[i][1] - a[j][1]) ** 2
                if dist <= m:
                    edges[i][j] = 1
        res = dfs(0, -1, edges, used, tin, up)
        flag = 0
        for i in range(n):
            if used[i] == 0:
                flag = 1
        if flag == 1 or res == 1:
            l = m
        else:
            r = m
    return str('%.6f' % (math.sqrt(r) / 2))
```

*Решение:*

Если радиус r удовлетворяет нашим условиям, то любой больший радиус тоже удовлетворяет. Поэтому мы можем воспользоваться двоичным поиском для поиска минимального подходящего радиуса. Для определения того, что при выходе из строя любого маяка, наша сеть все еще останется связной, можно воспользоваться методом поиска точек сочленения. Итоговая асимптотика: $O(N^2)$.

Пример программы, реализующей данный алгоритм на языке C++:

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cstdio>
#include <iomanip>
#include <vector>
#include <algorithm>

using namespace std;

typedef long long ll;

bool intersection(pair <int, int> &p1, pair <int, int> &p2, ll &r)
{
      ll dist = ((ll)p1.first - p2.first) * (p1.first - p2.first) +
      ((ll)p1.second - p2.second) * (p1.second - p2.second);
      return dist <= r;
}
```

```
int t;

bool dfs(int v, int p, vector < vector <int> > &edges,
         vector <bool> &used, vector <int> &in, vector <int> &up)
{
      used[v] = true;
      in[v] = up[v] = ++t;
      int child = 0;
      bool flag = false, was = false;
      for(auto i : edges[v]) {
            if(i == p) continue;
            if(!used[i]) {
                  child++;
                  was = max(was, dfs(i, v, edges, used, in, up));
                  up[v] = min(up[v], up[i]);
                  if(up[i] >= in[v]) flag = true;
            } else {
                  up[v] = min(up[v], in[i]);
            }
      }
      if(p == -1) flag = false;
      if(flag || (p == -1 && child > 1))
            return true;
      else
            return was;
}

int main() {
    int n;
    cin >> n;
    vector < pair <int, int> > v(n);
    for(auto &i : v) {
     cin >> i.first >> i.second;
    }
     ll l = -1, r = 2 * 1e18 + 1;
    while(r - l > 1) {
     ll m = (l + r) / 2;
     vector < vector <int> > edges(n);
     vector <bool> used(n, false);
     vector <int> in(n), up(n);
     t = 0;
     for(size_t i = 0; i < v.size(); i++) {
           for(size_t j = 0; j < i; j++) {
                 if(intersection(v[i], v[j], m)) {
                       edges[i].push_back(j);
                       edges[j].push_back(i);
                 }
           }
      }
      bool res = dfs(0, -1, edges, used, in, up);
      bool flag = false;
      for(auto i : used) {
           if(!i) flag = true;
      }
      if(flag || res)
           l = m;
```

```
      else
            r = m;
    }
    cout << fixed << setprecision(15) << sqrt((double)r) / 2;
      return 0;
}
```