Разбор задач

Максимальное количество баллов — 500

Задача 1. Почтовая марка

Рассмотрим верхнюю линию периметра. Она состоит из горизонтальных линий общей длины $w + 4$ и $w + 1$ вертикальных линий единичной длины. После упрощения получается следующее выражение: $2 \times w + 5$.

Рассмотрев аналогично другие линии периметра получаем общую формулу $4 \times w + 4 \times h + 20$.

Задача 2. Диалог нейросетей

Заметим, что в условии есть запрет на следование «poppush» — оно содержит «popp» и запрет на следование «inpush» — содержащее «npu». Отсюда следует, что окончание правильного диалога всегда будет иметь вид «offtoppush». Ещё запрещено повторение «poppop».

Остальные запреты касаются следования трёх слов подряд: запрещены «pushinpush», «pushinpop», «popinpop», «popinpush», «offtopinpush», «offtopinpop», «inpopofftop», «pushpopin».

Рассмотрим начало «pushpop». После этого нельзя поставить «in» из-за запрета «hpopi», остаётся добавить «offtop» и получить «pushpopofftop». Далее нужно добавить «in» и выйти на окончание «offtoppush», что даёт правильный диалог «pushpopofftopinofftoppush». Это один из самых коротких диалогов, он содержит минимальное число слов — 6 — и имеет длину 25 символов. Но из-за повторения длинного слова «offtop» — этот ответ не оптимален. Заметим, что «offtop» — единственное повторённое слово этом варианте диалога.

Начало «pushofftop» заведомо не может быть лучше, так как в дальнейшем мы снова должны будем использовать ещё одно вхождение «offtop» в окончании, а двойное вхождение «offtop» в ответ мы уже обсудили.

Теперь рассмотрим оптимальный вариант начала «pushin». Смысла добавлять далее «offtop» нет по причине того, что далее его придется добавлять ещё раз для выхода, поэтому желательно здесь поставить «pop». Напрямую этого делать нельзя из-за запрета «hinp». Но ничто не запрещает ещё раз повторить короткое слово «in» и избавиться от этого запрета: «pushininpop». Но теперь нельзя сразу добавить завершение «offtoppush» из-за запрета «npopo». Поэтому еще раз добавим слово «in» и только потом — «offtoppush». Получим самый короткий диалог «pushininpopinofftoppush». Он состоит из семи слов и имеет длину 23 символа.

Вот ещё варианты правильных диалогов из 25 символов: «pushofftoppopinofftoppush» и «pushinofftoppopofftoppush» — они так же состоят из шести слов. Остальные правильные диалоги имеют длину не менее 27 символов.

Задача 3. Робот-пылесос

Решение основывается на переборе разных вариантов маршрутов, где мы стремимся набрать как можно больше пыли.

Маршруты легче искать в такой таблице, если закрасить сектора в разные цвета в зависимости от количества пыли. Это легко можно сделать в электронных таблицах: нужно переписать данные в таблицу, выделить её и применить «Условное форматирование» –> «Цветовые шкалы» –> «Цветовая шкала зеленый-жёлтый-красный». Теперь маленькие числа будут красными, а большие — зелёными. Такая таблица называется тепловой картой.

	1	2	3	4	5	6	7	8	9
1	3	4	1	3	0	3	2	1	6
2	3	0	1	2	1	2	2	1	1
3	4	3	1		3	0	0	4	3
4	1	1	0	5	1	2	2	2	3
5	2	3	0	1	0	2	2	4	1
6	4	1	4	1	0	3	3	0	1
7	2	3	2	2	1	4	2	0	9

Тепловая карта помещения

Найдём решение для $X = 3$:

В радиусе трёх секторов от робота-пылесоса самые большие числа — это 5, 4 и несколько троек. Попытаемся их объединить и найти маршрут, который позволит роботу собрать наибольшее количество пыли.

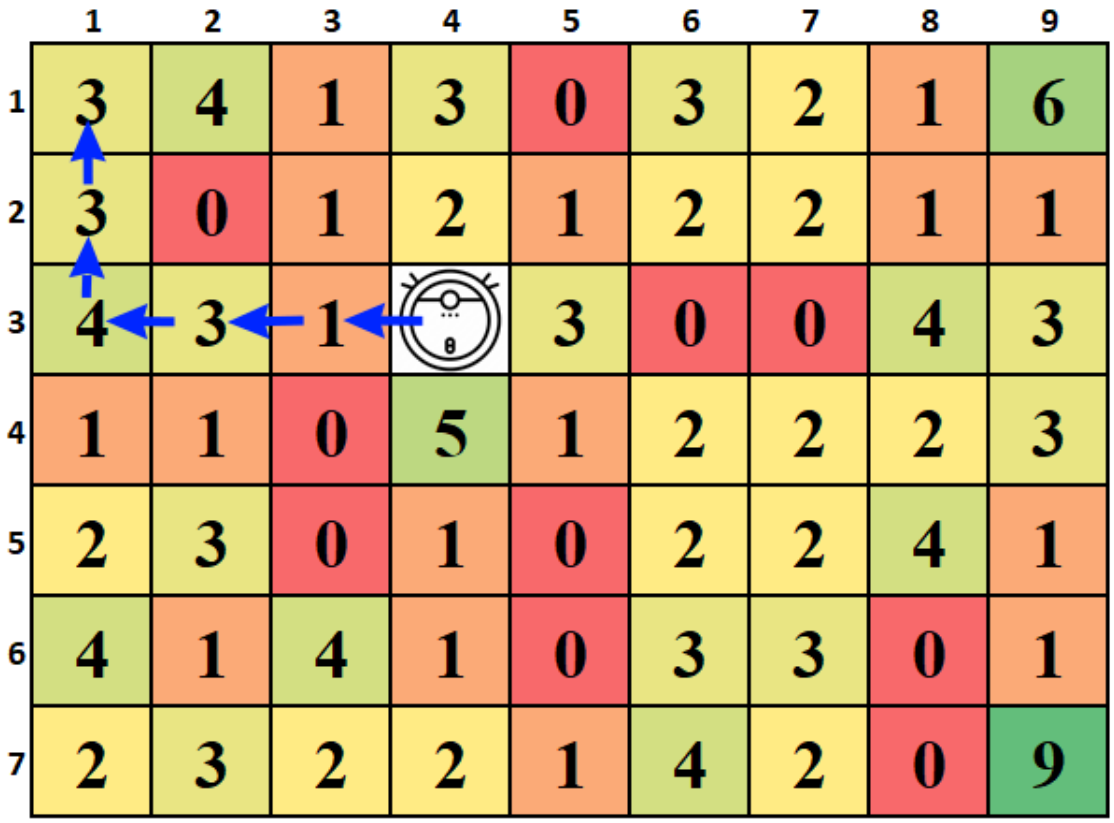
1. LLL 1+3+4=8
2. DRU 5+1+3=9
3. RDL 3+1+5=9

Остальные маршруты позволят собрать намного меньше пыли То есть наилучший маршрут позволяет собрать 9 единиц пыли и будет иметь вид «DRU» или «RDL». Пример маршрута «RDL»:

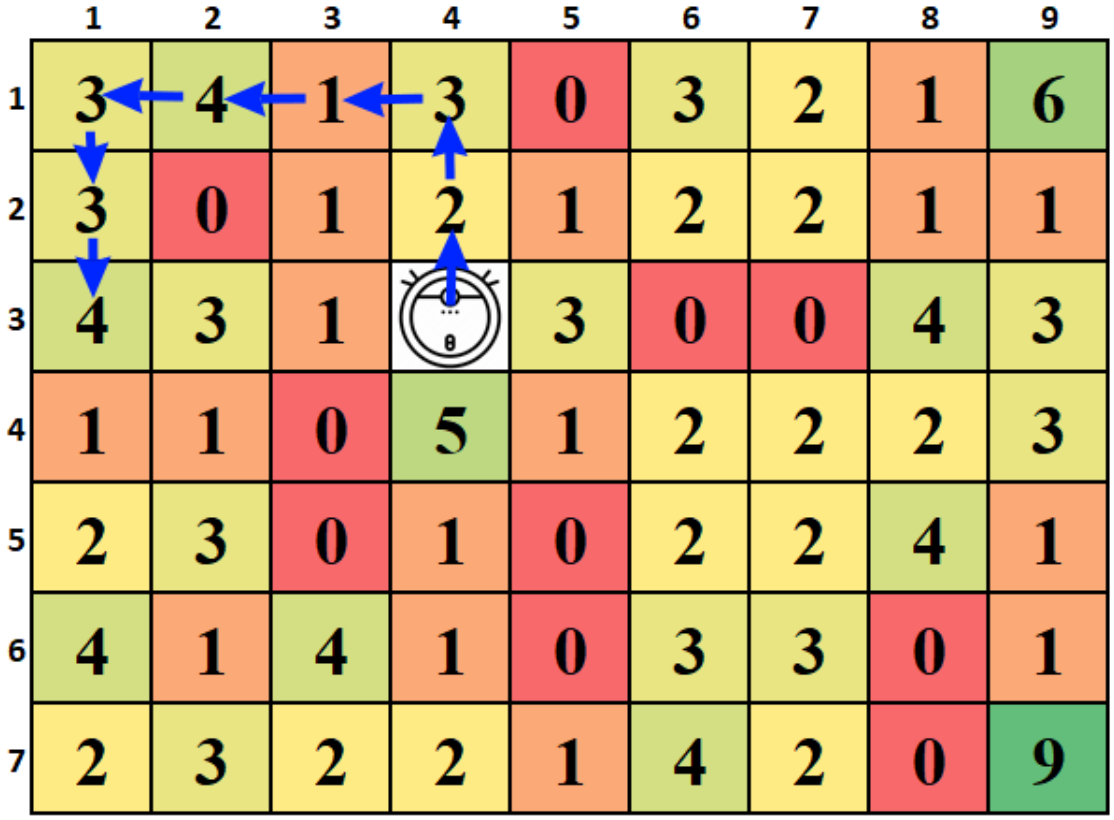
	1	2	3	4	5	6	7	8	9
1	3	4	1	3	0	3	2	1	6
2	3	0	1	2	1	2	2	1	1
3	4	3	1		3	0	0	4	3
4	1	1	0	5	1	2	2	2	3
5	2	3	0	1	0	2	2	4	1
6	4	1	4	1	0	3	3	0	1
7	2	3	2	2	1	4	2	0	9

Для нахождения ответа при $X = 5, 7, 9$ используем аналогичную логику. Определяем области секторов, где мы можем набрать больше всего пыли, и строим маршрут туда через сектора с наибольшими числами.

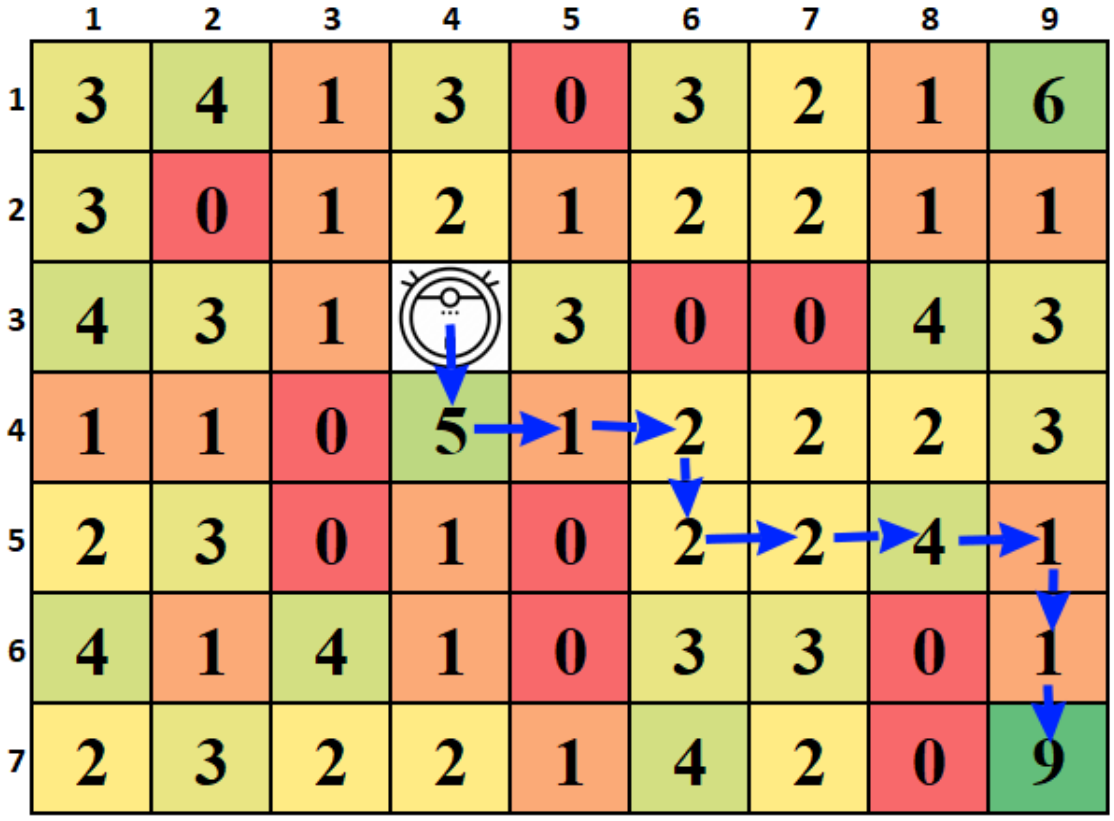
Для $X = 5$ маршрут «LLLUU» позволяет собрать 14 единиц пыли.



Для $X = 7$ маршрут «UULLLDD» позволяет собрать 20 единиц пыли.



Для $X = 9$ маршрут «DRRDRRRDD» позволяет собрать 27 единиц пыли.



Ещё один способ решения — написать программу, которая переберёт все маршруты нужной длины и найдёт маршрут, позволяющий собрать больше всего пыли. Полный перебор можно ре-

ализовать через рекурсивный алгоритм поиска в глубину. Такое решение в данной задаче будет работать довольно быстро, потому что маршрут максимальной длины не очень длинный.

```
x, y = 2, 3  # начальная координата робота
n, m = 7, 9  # размеры помещения

# карта помещения. Препятствия заменены -99,
# чтобы роботу было явно невыгодно туда ходить
field = [[3, 4, 1, 3, -99, 3, 2, 1, 6],
         [3, -99, 1, 2, 1, 2, 2, 1, 1],
         [4, 3, 1, 0, 3, -99, -99, 4, 3],
         [1, 1, -99, 5, 1, 2, 2, 2, 3],
         [2, 3, -99, 1, -99, 2, 2, 4, 1],
         [4, 1, 4, 1, -99, 3, 3, -99, 1],
         [2, 3, 2, 2, 1, 4, 2, -99, 9]]

# двумерный список, где мы будем отмечать сектора, по которым
# проехал робот-пылесос. 0 - не проехал, 1 - проехал
used = [[0] * m for _ in range(n)]

# из любого сектора можно проехать в одну из четырёх сторон
(список направлений).
# 0. y-1, x+0 - движение вверх (U)
# 1. y+1, x+0 - движение вниз (D)
# 2. y, x-1 - движение влево (L)
# 3. y, x+1 - движение вправо (R)
d = [[-1, 0], [1, 0], [0, -1], [0, 1]]

# функция, которая проверяет, находится ли робот-пылесос внутри помещения
def coord_ok(i, j):
    return 0 <= i < n and 0 <= j < m

# Функция рекурсивного перебора.
# i, j - текущая координата робота
# curEnergy - количество потраченного заряда
# curPoints - объём собранной пыли
# bp - путь, пройденный до текущей координаты
def dfs(i, j, curEnergy, curPoints, bp):
    # Если заряд закончился, заканчиваем движение
    if curEnergy == energy:
        return curPoints, bp
    maxPoints = 0
    bestPath = ""
    # отправим робота на 4 разные стороны и посмотрим,
    # откуда он принесет больше всего пыли.
    for k in range(4):
        i1 = i + d[k][0]
        j1 = j + d[k][1]
        if coord_ok(i1, j1):
            x = field[i1][j1]
            field[i1][j1] = 0
            points, path = dfs(i1, j1, curEnergy+1, curPoints+x, bp+str(k))
            if points > maxPoints:
                maxPoints = points
```

```
                bestPath = path
            field[i1][j1] = x
    return maxPoints, bestPath

# переберём длины маршрутов и для каждого случая найдём маршрут,
# который позволит роботу собрать наибольшее количество пыли.
for i in 3, 5, 7, 9:
    energy = i
    a, b = dfs(x, y, 0, 0, "")
    # заменим номера направлений на буквы.
    print(b.replace("0","U").replace("1","D").replace("2","L").replace("3","R"))
```

Задача 4. День борьбы

Рассмотрим несколько способов решения задачи.

Сначала посчитаем сумму всех чисел, например, используя формулу =SUM(A1:A1000). Эта сумма равна 74995. Значит, при оптимальном разбиении спортсменов на 3 группы, в каждой из трёх весовых категорий сумма весов должна оказаться примерно равной 25000.

Для каждой строки посчитаем сумму чисел в блоке от начала списка до этой строки (включительно). Для этого запишем в ячейку B2 формулу =SUM($A$1:A1), затем эту формулу скопируем в блок B1:B1000.

Аналогично для каждой строки посчитаем сумму чисел в блоке от конца списка до этой строки (включительно). Для этого запишем в ячейку C1000 формулу =SUM($A$1000:A1000), затем эту формулу скопируем в блок C1:C1000.

Попробуем найти в столбцах B и C значение, примерно равное 25000. В ячейке C685 находим число 25676, значит 316 последних участников в списке с весами от 78 и выше составляют тяжёлую весовую категорию с суммой весов, весьма близкой к оптимальной.

В столбце B есть два значения, близкие к искомому:

1) В ячейке B334 находим число 23058, значит, если первых 334 участников в списке с весами до 72 включительно объединить в лёгкую категорию, то в средней категории суммарный вес составит $74995 - 25676 - 23058 = 26261$. Это наибольшее число из трёх (23058, 25676 и 26261), и пока это лучшая из найденных прочностей помоста. При этом в средней весовой категории окажется $1000 - 316 - 334 = 350$ спортсменов.

2) В ячейке B398 находим число 27730, значит, участников с весами до 73 включительно можно объединить в лёгкую категорию. Однако их суммарный вес (27730) хуже, чем 26261, найденный нами в разборе предыдущего случая.

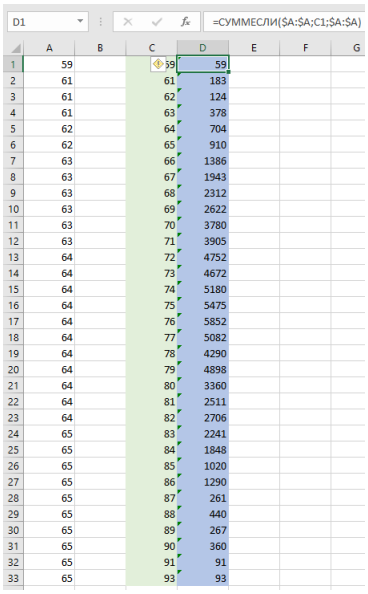
Перебором других близких вариантов можно убедиться, что это лучшее решение.

Ответ: 334, 350, 316.

Как можно было облегчить решение задачи?

С учётом того, что в списке много повторяющихся весов, можно сильно облегчить себе работу (и сократить обрабатываемые данные), если понять, что все значения у спортсменов одного и того же веса можно сложить в одно число — ведь их всё равно нельзя делить на части.

Создадим новый столбец с уникальными весами. Для этого скопируем столбец A в новое место (например, в столбец D) и избавимся от повторов (в MS EXCEL это можно сделать кнопкой «Удалить дубликаты» на вкладке «Данные». Останется всего 33 различных значений весов. Теперь просуммируем значения с одинаковыми весами и расположим их в соседнем столбце. Это можно сделать с помощью формулы =SUMIF($A:$A;D1;$A:$A), которую распространим на все соответствующие ячейки столбца E). Вот что должно получиться:



	A	B	C	D	E	F	G
1	59		59	59			
2	61		61	183			
3	61		62	124			
4	61		63	378			
5	62		64	704			
6	62		65	910			
7	63		66	1386			
8	63		67	1943			
9	63		68	2312			
10	63		69	2622			
11	63		70	3780			
12	63		71	3905			
13	64		72	4752			
14	64		73	4672			
15	64		74	5180			
16	64		75	5475			
17	64		76	5852			
18	64		77	5082			
19	64		78	4290			
20	64		79	4898			
21	64		80	3360			
22	64		81	2511			
23	64		82	2706			
24	65		83	2241			
25	65		84	1848			
26	65		85	1020			
27	65		86	1290			
28	65		87	261			
29	65		88	440			
30	65		89	267			
31	65		90	360			
32	65		91	91			
33	65		93	93			

Объём работы сократился с 1000 строк до 33.

Но можно и не создавать набор уникальных весов, а просто заметить, что веса принимают значения от 59 до 93. Давайте в некоторых ячейках запишем граничное значение массы спортсмена в этой категории. Например, в ячейках D1:D3 запишем числа 1, 2, 3, соответствующие номерам категорий, а в ячейках E1:E3 напишем максимальные значения массы спортсмена в этой категории. Теперь числа в блоке B1:B1000 заполним формулой, определяющей для каждого спортсмена номер категории: =IFS(A1<=$E$1;1;A1<=$E$2;2;A1<=$E$3;3)

Теперь в блоке F1:F3 посчитаем массу спортсменов соответствующей категории, например, при помощи формулы =SUMIF($B$1:$B$1000;D1;$A$1:$A$1000). А в блоке E1:E3 посчитаем количество спортсменов в этой категории, например, при помощи формулы =COUNTIF($B$1:$B$1000;D1).

Наконец, подберём такие граничные значения в блоке E1:E3, чтобы максимум в блоке F1:F3 оказался минимальным. Правильный ответ будет выглядеть так:

	A	B	C	D	E	F	G	H
1	59	1		1	72	23058	334	
2	61	1		2	77	26261	350	
3	61	1		3	93	25676	316	
4	61	1		Категория	Граница	Масса	Количество	
5	62	1						
6	62	1						
7	63	1						
8	63	1						
9	63	1						

Также можно написать программу на любом языке программирования. С учётом небольшого числа спортсменов (всего 1000) и большого числа повторяющихся весов, можно использовать полный перебор. Переберём все возможные количества спортсменов в лёгкой категории и для этого количества переберём все возможные количества спортсменов в средней категории. Если при этом числа на границах групп различны, определяем прочность помоста для каждой категории и выбираем из них наибольшее значение. Если это значение — наименьшее для всех найденных до этого момента, запоминаем его и соответствующие «границы» категории.

```
f = open("data.csv", "r")
Data = [int(x) for x in f]
n = 1000
best_max = 10 ** 18                 # Лучшее значение прочности помоста
for a in range(n - 2):              # Номер последнего спортсмена в лёгкой кат.
    for b in range(a+1, n-1):       # Номер последнего спортсмена в средней кат.
        if Data[a] != Data[a + 1] and Data[b] != Data[b + 1]:
            x = sum(Data[: a + 1])          # Сумма весов в лёгкой категории
            y = sum(Data[a + 1 : b + 1])    # Сумма весов в средней категории
```

```
            z = sum(Data[b + 1 :])          # Сумма весов в тяжёлой категории
            d = max(x, y, z)     # Прочность помоста (максимум из сумм весов)
            if d < best_delta:
                best_delta = d
                best_a = a+1    # Кол-во спортсменов в ответе в лёгкой кат.
                best_b = b-a    # Кол-во спортсменов в ответе в средней кат.
print(best_a)
print(best_b)
print(n  - best_a - best_b)
```

Задача 5. Обои и дипломы

Чтобы заполнить как можно большую площадь стены дипломами, Андрею нужно выкладывать их вплотную друг к другу, начиная с самого края стены, до тех пор, пока это возможно.

Для решения на 15 баллов Андрей может положить всего один диплом — ответом будет $a \cdot b$.

Для решения на 50 баллов можно смоделировать размещение дипломов на стене — прибавлять в переменную ширину диплома, пока она не превысит ширину стены, аналогично и с высотой.

Для полного решения необходимо вывести формулу. Чтобы закрыть стену дипломами полностью, например, в ширину, нужно, чтобы ширина стены делилась на ширину диплома. Если же она не делится, то остаток закрыть не получится. Таким образом, можно просто вычесть из размеров стены остатки от деления высоты стены n на высоту диплома a и ширины стены m на ширину диплома b и перемножить получившиеся результаты.

Пример решения на языке Python.

```
n = int(input())
m = int(input())
a = int(input())
b = int(input())
print((n - n % a) * (m - m % b))
```

Задача 6. Светофор

Минимальный номер такта, на котором машина проедет перекресток, можно найти как $\lceil n/b \rceil$, то есть частное с округлением вверх. Например, в Python частное с округлением вверх можно вычислить по формуле (n + b − 1) // b. Наибольшее число тактов будет достигаться, когда за один такт через перекрёсток проезжает минимальное число машин, то есть $\lceil n/a \rceil$.

Пример решения на языке Python.

```
a = int(input())
b = int(input())
n = int(input())
print((n + b - 1) // b)
print((n + a - 1) // a)
```

Задача 7. Робот

В решении на 30 баллов можно просто промоделировать движение робота, делая по одному шагу. В этом решении в переменной direction хранится значение +1 или −1, обозначающее изменение координаты при очередном шаге. Это значение меняется на противоположное (умножается на −1), когда количество шагов curr_steps, сделанных в данном направлении, станет равно величине max_steps, которая после этого увеличивается на 1.

```
n = int(input())
x = 0
direction = 1
curr_steps = 0
```

```
max_steps = 1
for i in range(n):
    x += direction
    curr_segment += 1
    if curr_steps == max_steps:
        direction *= -1
        curr_steps = 0
        max_steps += 1
print(x)
```

Чтобы улучшить это решение и набрать 65 баллов, будем моделировать перемещения не по одному шагу, а сразу добавляя к текущей координате 1, затем вычитая 2, добавляя 3 и т.д. Одновременно с этим будем считать количество оставшихся шагов, вычитая из значения n числа 1, 2, 3, пока значение n будет положительным. Поскольку на последнем отрезке может оказаться так, что мы сможем сделать не ровно steps шагов (переменная steps будет принимать значения 1, 2, 3, ...), а меньше, т.к. иначе n станет отрицательным, то будем вычитать не значение steps, а минимум из steps и n.

Пример решения на языке Python.

```
n = int(input())
direction = 1
steps = 1
x = 0
while n > 0:
    x += min(steps, n) * direction
    n -= min(steps, n)
    steps += 1
    direction *= -1
print(x)
```

Чтобы решить задачу на 100 баллов, необходимо быстро определить, сколько полных циклов из 1, 2, 3, ... шагов пройдёт робот. Пусть это значение равно p. Тогда нужно найти такое максимальное целое p, что $1+2+...+p \leqslant n$. Эту сумму можно вычислить по формуле арифметической прогрессии: $1 + 2 + ... + p = p(p + 1)/2$. Итого нам нужно найти такое максимальное целое p, что $p(p + 1) \leqslant 2n$. Вместо этого возьмём $p = \sqrt{2n}$, округлив вниз до целого. То есть мы возьмём такое целое p, что $p^2 \leqslant 2n$, но при этом может оказаться так, что $p(p + 1) > 2n$. Несложно понять, что мы можем ошибиться не более, чем на 1, поэтому проверим, не возникла ли ошибка, и уменьшим значение p при необходимости.

Если было выполнено p полных циклов, то робот сделал $p(p+1)/2$ шагов, поэтому ему осталось сделать ещё $n - p(p+1)/2$ шагов. Дальнейшие случаи зависят от того, будет ли значение p чётным или нечётным. После выполнения 1, 2, 3, 4, 5 и т.д. полных циклов координата робота будет равна 1, −1, 2, −2, 3 и т.д. То есть при нечётном p робот закончит цикл в клетке $(p + 1)/2$, а значение $n - p(p + 1)/2$ нужно будет вычесть. При чётном p робот закончит цикл в клетке $-p/2$, а значение $n - p(p + 1)/2$ нужно будет прибавить.

Пример решения на языке Python.

```
n = int(input())
p = int((2 * n) ** 0.5)
if p * (p + 1) > 2 * n:
    p -= 1
if p % 2 == 1:
    x = (p + 1) // 2
    x -= n - p * (p + 1) // 2
else:
    x = - p // 2
    x += n - p * (p + 1) // 2
```

```
print(x)
```

Также можно первую часть решения (нахождение наибольшего p такого, что $p(p+1) \leqslant 2n$) выполнить двоичным поиском. Пример такого решения.

```
n = int(input())
left = 0
right = n
while right - left > 1:
    mid = (left + right) // 2
    if mid * (mid + 1) <= 2 * n:
        left = mid
    else:
        right = mid
p = left
if p % 2 == 1:
    x = (p + 1) // 2
    x -= n - p * (p + 1) // 2
else:
    x = - p // 2
    x += n - p * (p + 1) // 2
print(x)
```