Всероссийская олимпиада школьников по информатике
Вологодская область, 2021-2022 учебный год
II (муниципальный) этап
7 - 8 классы
Методические рекомендации по разбору задач

Задача 1. Муравей и кубик (100 баллов)

Вопрос 1. Ответ равен 4, найти его не представляет сложности.

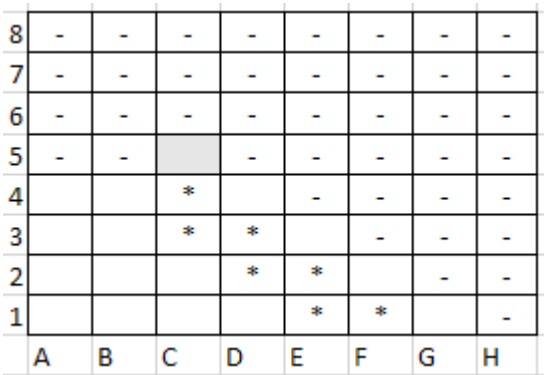
Вопрос 2**.** Существует 12 таких клеток – это как раз те клетки, через которые проходит Муравей.

Вопрос 3**.** Подходит любое положение кубика, кроме последней строки и последнего столбца. Командами в цикле Муравей сдвинет кубик в верхнюю строку, в каком бы столбце он ни находился. Далее командами после цикла сдвинет его в начало строки. Ответ равен 49.

Вопрос 4**.** Заметим, что начав двигаться из какой-то клетки, Муравей никогда не опустится ниже её (разве что кроме верхней строчки, но она нам не помешает). Поэтому можно сразу отбросить из рассмотрения все клетки выше кубика, а также в строке с кубиком.

Также можно заметить, что после каждого выполнения цикла Муравей оказывается на одну клетку левее и на одну выше предыдущего положения. При этом он проходит ещё через 4 клетки выше и правее. По сути, Муравей движется по линии, параллельной побочной диагонали, и при этом захватывает ещё две соседних таких линии выше. Поэтому подходящие клетки не могут быть выше линии, параллельной побочной диагонали и проходящей через кубик.

Этими рассуждениями мы отбросили большое число заведомо неподходящих клеток – на рисунке они обозначены прочерками). Оставшиеся клетки уже несложно проверить вручную (часть из них тоже быстро отбрасывается). Подходящие клетки на рисунке обозначены звёздочками.



Всего получилось 7 клеток, стартовав из которых, Муравей доставит кубик в клетку A8.

Также можно было найти ответ на данный вопрос, написав программу, моделирующую движение Муравья.

Все ответы к задаче: 4 12 49 7

Задача 2. Красота степеней (100 баллов)

Вопрос 1. Последняя цифра повторяется с периодом 4. Значит, у любых двух степеней 7^x и 7^y последние цифры равны, если показатели *x* и *y* имеют одинаковые остатки от деления на 4. Поскольку 190 *mod* 4 = 2, то последняя цифра у 7^{190} такая же, как у 7^2 – это цифра 9.

Также можно было сосчитать ответ на Python (или Java с типом BigInteger).

Вопрос 2**.** Заметим, что при перемножении двух чисел последние две цифры результата определяются только последними двумя цифрами множителей, а остальные цифры множителей можно стереть. Поэтому у числа $7070707^{7070707}$ последние две цифры будут такими же, как и у числа $07^{7070707}$, оно же $7^{7070707}$.

Из таблички в условии задачи видно, что последние две цифры у степеней семёрки повторяются с периодом 4. Поскольку 7070707 *mod* 4 = 3, то последние две цифры у $7^{7070707}$ такие же, как у 7^3 – ответ равен 43.

Участники, знающие встроенные математические функции языка Python, могли легко сосчитать ответ так:

pow(7070707, 7070707, 100)

Вопрос 3**.** Преобразуем выражение:

$7^{123456789} + 7^x = 7^x * (7^{123456789-x} + 1)$

Понятно, что левый множитель на 11 не делится. Чтобы правый множитель делился на 11, нужно, чтобы выполнялось: $7^{123456789-x}$ *mod* 11 = 10. Посмотрим, как чередуются остатки от деления на 11 для степеней семёрки:

7^x	7^1	7^2	7^3	7^4	7^5	7^6	7^7	7^8	7^9	7^{10}	7^{11}
7^x mod 11	7	5	2	3	10	4	6	9	8	1	7

Начиная с 7^{11}, остатки начали повторяться – то есть, период равен 10. Нам нужен остаток 10 – его дают степени 7^5, 7^{15}, 7^{25}, и так далее. Наименьшее *x*, при котором получается показатель степени с пятёркой на конце – это 4. Ответ равен 4.

На языке Python ответ можно было получить так: вычисляем выражение pow(7, 123456789, 11) + pow(7, x, 11) для x = 1, 2, 3, 4 … и смотрим, при каком x результат будет делиться на 11 – это случится при x=4.

Вопрос 4**.** Найти ответ на данный вопрос, пользуясь только бумагой и ручкой, наверное, не очень просто. Проще всего написать небольшую программу, которая подберёт нужное *x*. Пример такой программы на Python:

```
for x in range(1, 1000):
   if (pow(7, x, 100) + pow(x, 7, 100)) % 100 == 21:
       print(x)
       break
```

На языках без столь удобной функции, как *pow* в Python, и без встроенной поддержки длинной арифметики это тоже можно сделать, хотя и чуть сложнее. Пример решения на Pascal:

```
var
   x, i, p7x, px7: longint;
begin
    for x := 1 to 1000 do begin
      p7x := 1;
      for i := 1 to x do
         p7x := p7x * 7 mod 100;
      px7 := 1;
      for i := 1 to 7 do
         px7 := px7 * x mod 100;
      if (px7 + p7x) mod 100 = 21 then begin
          writeln(x);
          break;
      end;
    end;
end.
```

Участники, которые программировать не умеют совсем, могли бы решить эту задачу с использованием электронных таблиц. Вариант решения в Excel может выглядеть так. В столбец A записываем числа 1 и 2, выделяем их и протягиваем мышкой вниз – чтобы получились последовательные числа до 100 (можно больше).

В верхнюю ячейку столбца B записываем число 7, в ячейку ниже пишем формулу “=ОСТАТ(B1*7;100)”, протягиваем эту формулу вниз на 100 ячеек. В столбце должны получиться числа 7, 49, 43 и так далее – остатки от деления чисел 7^1, 7^2, 7^3 … на 100.

В верхнюю ячейку столбца C записываем формулу “=ОСТАТ(СТЕПЕНЬ(A1;7);100)” и протягиваем её вниз на 100 ячеек.

В столбце должны получиться числа 1, 28, 87 и так далее – это остатки от деления 1^7, 2^7, 3^7 … на 100.

Суммируем числа в столбцах B и C – для этого в верхнюю ячейку столбца D записываем формулу “=СУММ(B1:C1)” и протягиваем её вниз на 100 ячеек. Начало таблицы выглядит так:

	A	B	C	D
1	1	7	1	8
2	2	49	28	77
3	3	43	87	130
4	4	1	84	85
5	5	7	25	32
6	6	49	36	85
7	7	43	43	86
8	8	1	52	53
9	9	7	69	76

Осталось только найти в столбце D число, которое оканчивается на 21, при этом значение в столбце A будет ответом. Ответ будет равен 98.

Ответы на все вопросы задачи: 9 43 4 98

Задача 3. Клонирование машин (100 баллов)

После первого клонирования мы имеем две готовые машины, после второго – 4, после третьего – 8, и так далее. Таким образом, нужно найти минимальное натуральное x, при котором $2^x \geq N$, а это будет логарифм от N по основанию 2 с округлением вверх. Пример решения на Python в две строчки кода:

```
import math
print(math.ceil(math.log(int(input()), 2)))
```

Альтернативное решение – подобрать в цикле показатель степени:

```
n = int(input())
ans = 0
count = 1
while count < n:
    count *= 2
    ans += 1
print(ans)
```

Задача 4. Чётные суммы (100 баллов)

Первую подзадачу можно решить «в лоб» с помощью вложенных циклов.

Во второй подзадаче можно заранее сосчитать количество чётных и нечётных чисел на интервале. Далее в цикле перебираем первый элемент пары и добавляем к ответу подходящее количество вторых элементов (чётных, если первый элемент чётный, и наоборот).

Для решения третьей подзадачи вначале заметим, что количество чётных чисел от 1 до *X* равно *X* div 2, а нечётных – (*X*+1) div 2. Тогда количество чётных чисел в интервале от *A* до *B* равно:

even = B div 2 - (A - 1) div 2

Количество нечётных чисел в интервале от *A* до *B* равно:

odd = (B+1) div 2 - A div 2

Пара с чётной суммой может состоять либо из двух чётных чисел, либо двух нечётных. Количество пар из чётных чисел равно even*(even-1)/2, из нечётных – odd*(odd-1)/2. Итого ответ равен:

(even * (even - 1) + odd * (odd - 1)) div 2

Пример решения на языке Python:

```
a = int(input())
b = int(input())
even = b // 2 - (a - 1) // 2
odd = (b + 1) // 2 - a // 2
print((even * (even - 1) + odd * (odd - 1)) // 2)
```

Задача 5. Факториал и степень (100 баллов)

Первую подзадачу можно решить «в лоб»: циклами сосчитать факториал и степень, проверить делимость.

Во второй подзадаче можно использовать ту же идею, что и в третьей (смотрите далее), но реализовать её менее эффективно – например, искать простые множители циклом не до корня из числа, а до самого числа, считать их количество в факториале циклом до *N*.

Участники, пишущие на Питоне, могли также решить вторую подзадачу с использованием функции *pow* и функции *factorial* из модуля *math*, которые работают достаточно быстро. При этом попытка вычислить факториал вручную в цикле (без каких-либо оптимизаций) должна давать предел времени.

Полное решение задачи выглядит так. Разобьём число *a* на простые множители, используя цикл до корня квадратного из *a*.

Пусть p – очередной простой множитель, x – сколько раз он встречается в числе a. Тогда в числе a^b этот множитель встречается $x \cdot b$ раз.

Найдём теперь, сколько раз он встречается в факториале. Количество чисел от 1 до N, которые делятся на p, равно N div p. Но в некоторых числах множитель встречался дважды – такие числа делятся на p^2, а их количество равно N div p^2. Аналогичные рассуждения для p^3, и так далее. Таким образом, в числе $N!$ простой множитель p встречается следующее число раз:

$$N \text{ div } p + N \text{ div } p^2 + N \text{ div } p^3 + \ldots$$

Ряд обрывается, когда очередной член будет равен нулю.

Осталось проверить: если в $N!$ какой-то простой множитель p содержится меньшее число раз, чем в a^b, то ответ будет “No”. Если же каждый множитель в факториале содержится не меньше раз, чем в степени, то ответ – “Yes”.