Разбор задач

Задача 1. Отпуск

Решение с помощью цикла позволяет набрать 60 баллов: будем перебирать номера дней в цикле и считать количество воскресений, начиная с первого понедельника.

```
n = int(input())
d = int(input())

monday = False
sunday = 0

for i in range(n):
    if (d + i) % 7 == 1:
        monday = True
    elif (d + i) % 7 == 0 and monday:
        sunday += 1
print(sunday)
```

Рассмотрим идею эффективного решения. Заметим, что если $d = 1$ (первый день отпуска — понедельник), то ответом на задачу будет $n//7$ - количество полных недель в отпуске.

Для случая, когда d любое определим, сколько дней отпуска Иван Петрович вынужден будет провести в родном городе до вылета на Кипр. В большинстве случаев это есть количество дней с начала отпуска до следующего за ним понедельника, то есть $(8 - d)$. Но если начало отпуска выпадает на понедельник, то нам нужно получить 0, а приведённая формула даёт 7, поэтому скорректируем её следующим образом: $(8-d) \bmod 7$, где $\bmod$ — это операция взятия остатка.

Теперь для всех возможных d это есть количество дней отпуска, которые Иван Петрович проведёт дома до вылета на Кипр. После этого у него от отпуска останется ещё $\max(0, n - (8 - d) \bmod 7)$ дней. Поскольку из этого периода на Кипре он проведёт некоторое целое количество недель с понедельника по воскресенье, то получаем окончательный ответ на вопрос задачи:

$$\left[\frac{\max(0, n - (8 - d) \bmod 7)}{7}\right],$$

где $[\cdot]$ – целая часть числа.

Ниже приведено решение на языке Python.

```
n = int(input())
d = int(input())

print(max(0, (n - (8 - d) % 7) // 7))
```

Задача 2. Гирьки

Если количество гирек весом 1 (грамм) нечётно, то общий весь всех гирек нечётен и разложить на кучки равного веса нельзя. Если гирек веса 1 вообще нет, то разложить гирьки на 2 кучки равного веса можно тогда и только тогда, когда количество гирек веса 2 чётно — в этом случае в каждую из кучек нужно положить половину всех гирек веса 2.

Во всех остальных случаях гирьки разложить можно: например, в одну из кучек можно положить гирьки веса 2 в таком максимально возможном количестве, чтобы общий вес этой кучки не

превосходил половину веса всех гирек, и затем, если потребуется, добавить гирьки веса 1 до нужного веса. Более формально: пусть $w = (n_1 + 2 \cdot n_2)/2$ – половина суммарного веса всех гирек. Тогда положим:

$$k_2 = \min\left(n_2, \left[\frac{w}{2}\right]\right),$$

$$k_1 = w - 2 \cdot k_2,$$

где $[\cdot]$ – целая часть числа, и возьмём в одну из кучек k_1 гирек веса 1 и k_2 гирек веса 2.

Ниже представлено верное решение на языке Python

```
n1 = int(input())
n2 = int(input())

if (n1 % 2 != 0) or (n1 == 0 and n2 % 2 != 0):
    print("No")
else:
    print("Yes")
    w = n1 // 2 + n2
    k2 = min(n2, w // 2)
    k1 = w - k2 * 2
    print(k1, k2)
```

Задача 3. Конструктор

Если имеется ровно k брусков длины a, то из этих брусков можно собрать $\left[\frac{k}{3}\right]$ равносторонних треугольников. Поэтому для решения задачи для каждой последовательности подряд идущих одинаковых элементов нужно вычислить её длину и просуммировать указанную величину по всем таким последовательностям.

Пример верного решения на языке Python, не требующего сохранения данных в списке или других специальных структурах:

```
n = int(input())
c = 0
left = 0
x = int(input())

for right in range(1, n):
    y = int(input())
    if x != y:
        c += (right - left) // 3
        x = y
        left = right

print(c + (n - left) // 3)
```

Концептуально более простое полное решение можно получить с использованием структуры данных «словарь» — подсчитаем количество каждого бруска, а затем пройдем по этим значениям и получим ответ:

```
c = dict()
n = int(input())
```

```
for i in range(n):
    a = int(input())
    c[a] = c.get(a, 0) + 1

ans = 0
for d in c.values():
    ans += d // 3
print(ans)
```

Задача 4. Обучение шахматам

Заметим, что слон ходит только по клеткам одного и того же цвета, причем он может попасть на любую клетку данного цвета. Действительно, легко видеть, что слон, который ходит по белым клеткам, из любой клетки может попасть в белый угол доски, и, соответственно, из белого угла может попасть в любую клетку. Таким образом из любой белой клетки до любой другой можно добраться через белый угол. Аналогично из любой клетки до любой другой можно добраться через чёрный угол. Значит, чтобы понять, может ли слон из исходной клетки (x_1, y_1) попасть в клетку (x_2, y_1), нужно понять, одного они цвета или нет. Нетрудно видеть, что цвет клеток определяется чётностью суммы координат. То есть слон может из первой клетки попасть во вторую тогда и только тогда, когда $x_1 + y_1$ и $x_2 + y_2$ имеют одинаковую четность. Или, что то же самое, когда сумма всех четырёх чисел чётна:

$$(x_1 + y_1 + x_2 + y_2) \mod 2 = 0.$$

Рассмотрим несколько способов решения данной задачи на полный балл — поиск подходящего маршрута шахматного слона.

1. Чтобы построить путь слона из одной точки в другую можно заметить следующее: диагонали, идущие в одном направлении, сохраняют величину суммы $x + y$ для всех клеток диагонали, а идущие в перпендикулярном — величину разности $x - y$. Поэтому, если

   $$x_1 + y_1 = x_2 + y_2$$

   или

   $$x_1 - y_1 = x_2 - y_2,$$

   то начальная и конечная клетка лежат на одной диагонали и попасть из одной в другую можно за один ход.

   Если же ни одно из этих равенств не выполнено, то рассмотрим 2 случая. Первый: $x_1 + y_1$ — четно. В этом случае пройдём из начальной клетки в конечную через главную диагональ, задающуюся уравнением $x - y = 0$. Из клетки (x_1, y_1) первым ходом перейдём в клетку (X_1, Y_1), лежащую одновременно с ней на одной диагонали и на главной диагонали:

   $$X_1 = Y_1,\ X_1 + Y_1 = x_1 + y_1 \Rightarrow$$

   $$X_1 = Y_1 = \frac{x_1 + y_1}{2}.$$

   Если же клетка (x_1, y_1) сама лежит на главной диагонали, то этот ход пропускается и сразу переходим в следующую клетку. Следующая клетка определяется аналогично:

   $$X_2 = Y_2 = \frac{x_2 + y_2}{2}.$$

   Если (x_2, y_2) сама лежит на главной диагонали, то цель достигнута, иначе перейдем из клетки (X_2, Y_2) в конечную клетку.

   Второй случай: $x_1 + y_1$ — нечетно. В этом случае нужно проделать аналогичные перемещения через побочную диагональ, задающуюся уравнением $x + y = 9$ (нумерация строк и столбцов считается с 1).

2. Другой способ решения заключается в постепенном приближении к клетке (x_2, y_2).

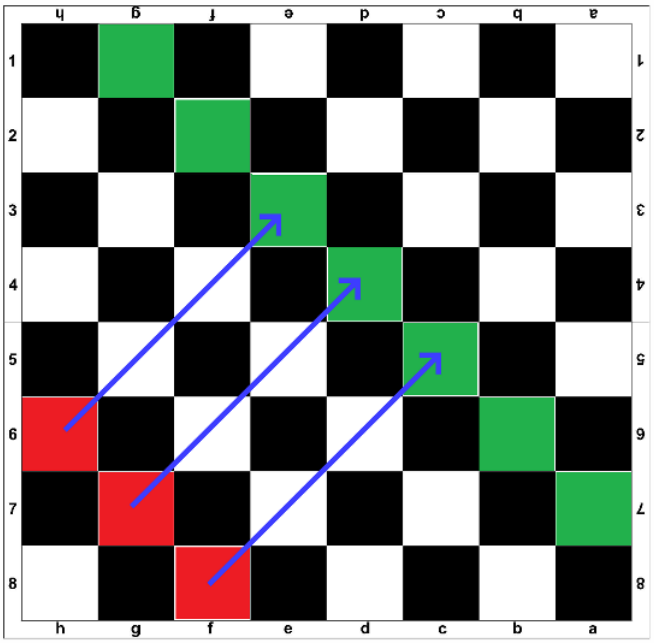
   Для этого на каждом шаге будем делать один ход в ее направлении, уменьшая или увеличивая координаты текущей клетки на 1. Код на языке Python представлен ниже:

```
x1, y1 = int(input()), int(input())
x2, y2 = int(input()), int(input())

def moveto(a, b):
    if a > b:
        return a - 1
    elif a < b:
        return a + 1
    elif a == 1:
        return 2
    else:
        return a - 1

if (x1 + y1 + x2 + y2) % 2 != 0:
    print("No")
else:
    ans = []
    while x1 != x2 or y1 != y2:
        x1 = moveto(x1, x2)
        y1 = moveto(y1, y2)
        ans.append(str(x1) + " " + str(y1))
    print("Yes")
    print(len(ans))
    print(*ans, sep="\n")
```

3. Заметим, что если маршрут существует, то слону потребуется не более 2 ходов: если начальная и конечная клетка находятся на одной диагонали, то достаточно одного хода. Если на разных диагоналях, то с короткой диагонали всегда можно попасть на нужную длинную диагональ за один ход слона, после чего сделать ровно один ход по этой диагонали. Очевидно, что если нужно попасть с длинной на короткую, то нужно проделать те же ходы в обратном порядке.



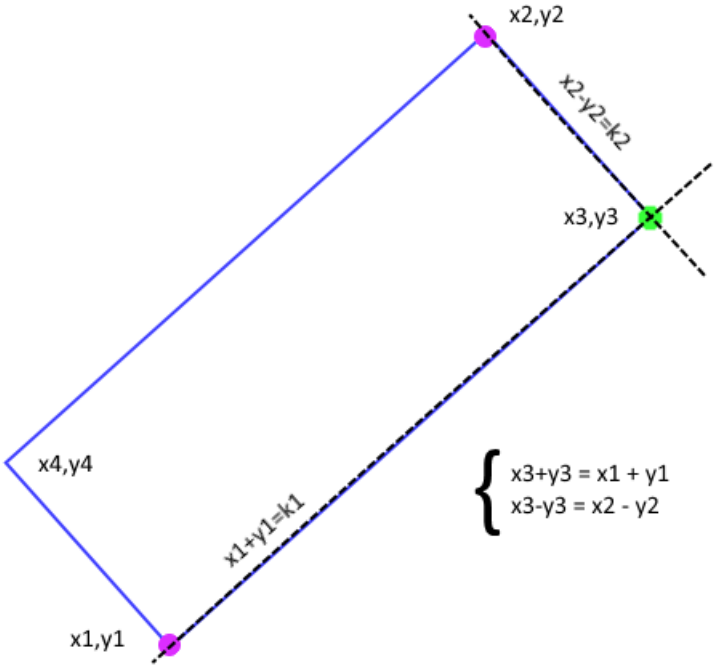
Таким образом, нам достаточно перебором найти клетку шахматного поля, из которой можно попасть ходом слона как в начальную, так и в конечную клетки. Код на языке Python приведен ниже:

```
x1, y1 = int(input()), int(input())
x2, y2 = int(input()), int(input())

if (x1 + y1) % 2 != (x2 + y2) % 2:
    print("No")
else:
    print("Yes")
    print(2)
    for i in range(64):
        x3, y3 = i // 8 + 1, i % 8 + 1
        if abs(x1 - x3) == abs(y1 - y3) and abs(x2 - x3) == abs(y2 - y3):
            print(x3, y3)
            break
    print(x2, y2)
```

4. Представим прямоугольник, у которого начальная (x_1, y_1) и конечная (x_2, y_2) точки маршрута слона — это противоположные точки на одной из диагоналей прямоугольника. Тогда концы другой диагонали (x_3, y_3) и (x_4, y_4) — это точки, через которые слон может попасть из начальной в конечную точку. При этом хотя бы одна из этих точек находится на доске, исходя из предыдущего пункта.

   Тогда для того, чтобы найти координаты точки, через которую нужно выполнить переход достаточно найти точки, являющиеся концами противоположной диагонали. Как найти одну из таких точек — показано на рисунке ниже:



Если найденная точка лежит вне шахматной доски, то найдем другую точку аналогичным способом.

Код на языке Python представлен ниже:

```
x1, y1 = int(input()), int(input())
x2, y2 = int(input()), int(input())

if (x1 + y1) % 2 != (x2 + y2) % 2:
    print("No")
else:
    print("Yes")
    x3 = (x1 + y1 + x2 - y2) // 2
    y3 = (x1 + y1 - x2 + y2) // 2
    if max(x3, y3) > 8 or min(x3, y3) < 1:
        x3 = (x2 + y2 + x1 - y1) // 2
        y3 = (x2 + y2 - x1 + y1) // 2
    print(2)
    print(x3, y3)
    print(x2, y2)
```

Задача 5. Интересные числа

Рассмотрим несколько вариантов решения данной задачи.

Первый (медленный) способ заключается в переборе подряд всех чисел и поиске для каждого из них максимального простого делителя. Если такой делитель не будет превосходить 5, то данное число подходит. Будем перебирать числа, начиная с n до 1 пока не найдем подходящее число. Такое решение позволяет набрать 30 баллов.

```
def isSuit(x):
    ans = 1
```

```
    p = 2
    while p * p <= x:
        if x % p == 0:
            ans = p
            while x % p == 0:
                x = x // p
        p += 1
    if x > 1:
        ans = x
    return ans <= 5

n = int(input())
while not isSuit(n):
    n -= 1
print(n)
```

Еще одно решение можно получить, перебирая подряд все числа и выполняя деление на 2, 3 и 5. Если в итоге остается 1, это значит, что число нам подходит, так как других простых делителей у числа не было. Такое решение позволяет набрать 50 баллов.

```
def isSuitable(x):
    for p in 2, 3, 5:
        while x % p == 0:
            x //= p
    return x == 1

n = int(input())
while not isSuitable(n):
    n -= 1
print(n)
```

Переберём в цикле все числа вида $2^i \cdot 3^j \cdot 5^k$, не превосходящие n — их совсем немного: из условия $n \leqslant 10^{17}$ следует $0 \leqslant i \leqslant 56$, $0 \leqslant j \leqslant 35$, $0 \leqslant k \leqslant 24$. И среди этих чисел выберем наибольшее — оно и будет ответом в задаче.

```
def solve():
    n = int(input())
    ans = 2
    p1 = 1
    while p1 <= n:
        p2 = 1
        while p1 * p2 <= n:
            p3 = 1
            while p1 * p2 * p3 <= n:
                if p1 * p2 * p3 > ans:
                    ans = p1 * p2 * p3
                p3 *= 5
            p2 *= 3
        p1 *= 2
    return ans

print(solve())
```

Стоит отметить, что если просто перебирать все числа указанного вида в указанном диапазоне, то возникнет переполнение целочисленных значений и программы могут давать неправильные ответы. Поэтому нужно перебирать постепенно повышая одну из степеней в разложении числа — в этом случае благодаря ограничению $n \leqslant 10^{17}$ получаемые значения не выйдут за диапазон 64-битных целых чисел в языках, где такое переполнение возможно.

Последовательность “интересных” чисел в информатике также называется последовательностью Хэмминга.