Разбор задач

Задача 1. Конфеты

Для нахождения минимального ответа будем считать, что при каждом делении конфет было поровну, для нахождения максимального — считаем, что Алина оставляла у себя при каждом делении на одну конфету меньше.

В случае перебора вариантов понадобится рассмотреть только 8 различных вариантов, так как делений было ровно 3.

Ответ:
8 15
56 63
120 127

Задача 2. Предатель

Black и *Red* допущены к управлению космическим кораблем, но при этом мы знаем, что только инженер и полицейский допущены к управлению космическим кораблем, значит среди них инженер и полицейский, тогда как предатель и медик среди *Green* и *White*.

Так как инженер встречал сегодня только полицейского, значит он не мог обыграть *White* в кают-компании, то есть *Black* не инженер. Значит инженер - это *Red*, а полицейский — это *Black*.

Известно, что полицейский и предатель старше инженера. Так как, инженер — это *Red* и он старше *Green*, значит *Green* — это медик, а *White* — предатель.

Ответ: $WBRG$

Задача 3. Асфальт

Для вывода формулы нужно посчитать 3 слагаемых:

1. площадь асфальтового покрытия между жилыми массивами в каждом вертикальном ряду (за исключением перекрестков): m рядов по $(n-1)$ отрезков дороги размером k на L
2. площадь асфальтового покрытия между жилыми массивами в каждом горизонтальном ряду (аналогично предыдущему)
3. площадь перекрестков: $(n-1)*(m-1)*L*L$

Сложив эти 3 слагаемых получим искомое значение:

n	m	L	k	Ответ
3	3	5	10	700
2	5	10	20	3000
10	10	15	30	99225
10	20	30	40	597900

Задача 4. В гостях у Гены

Решение задачи подразумевает перебор вариантов. Так как Гена всегда сидит на первом месте, а Зоя должна быть рядом, то возможны всего два их взаимных расположения:

1) $G----Z$

2) $GZ----$

На второе свободное рядом с Зоей место Гена посадит Игоря или Бориса, так как другие варианты для него нежелательны по условию. Получаем 4 варианта:

1) $G---IZ$

2) $GZI--$

3) $G---BZ$

4) $GZB--$

Для каждого случая получим все возможные рассадки (так, чтобы Дима не был рядом с Игорем):
1) $G---IZ$: $GDFBIZ$, $GDBFIZ$, $GFDBIZ$, $GBDFIZ$

2) $GZI--$: $GZIBDF$, $GZIFDB$, $GZIBFD$, $GZIFBD$

3) $G---BZ$: $GDFIBZ$, $GIFDBZ$

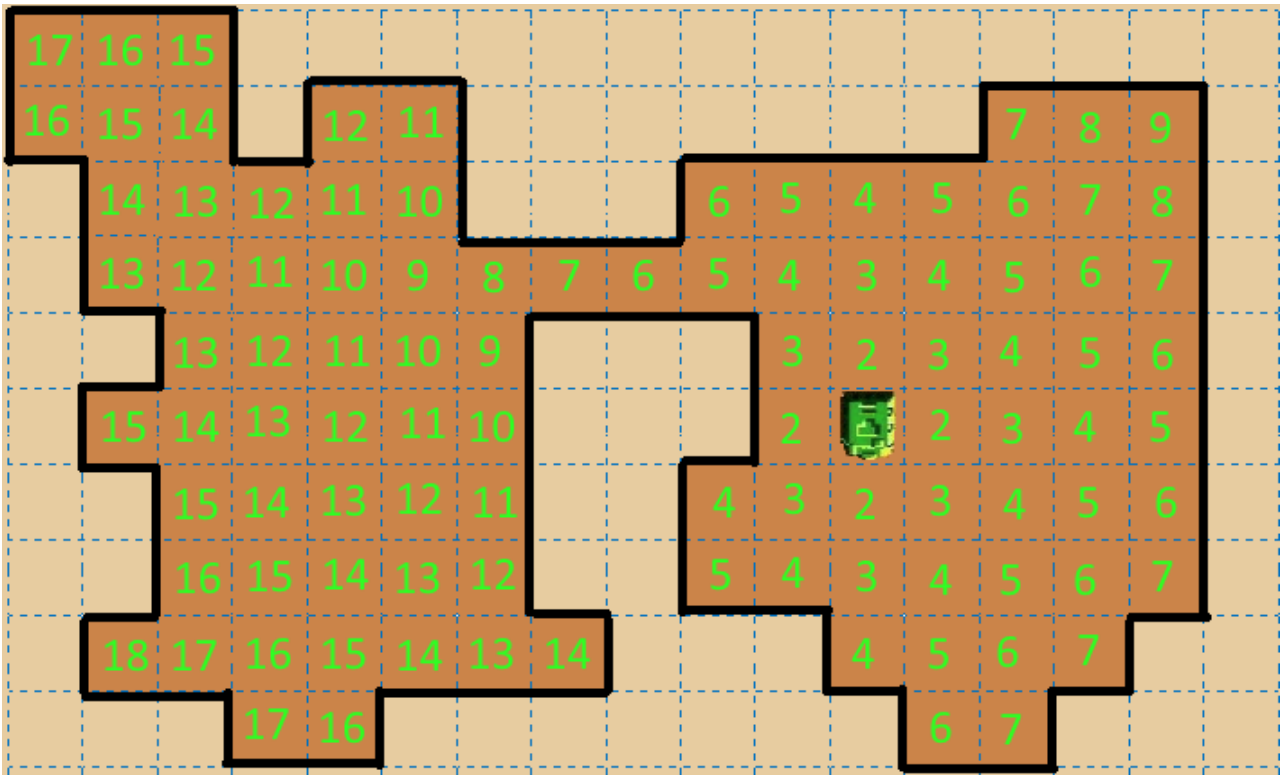
4) $GZB--$: $GZBDFI$, $GZBIFD$

Всего получаем 12 допустимых вариантов рассадки, из которых нужно выбрать и записать в ответ любые 10

Задача 5. Арракис

Для того, чтобы решить эту задачу будем отмечать клетки, до которых харвестер может добраться числами. Помечаем начальную клетку 1, дальше все соседние с ней в перекрестии клетки, в которых еще не стоят числа помечаем 2 и так далее. В итоге для каждого значения нужно посчитать количество клеток с величиной большей, чем величина в клетке.

Для первых 3 чисел проще посчитать количество клеток, которые харвестер сможет достать и вычесть эти значения из общего количества (100). Для последнего значения - проще посчитать количество клеток, которые останутся.



Ответ: 88 68 52 4

Разбор задач

Задача 1. Творческая натура

Первый ряд плитки займет места: $k + n + k$. Каждый следующий ряд плитки «включается» в предыдущий ряд на глубину k, поэтому каждый следующий ряд добавляет $n + k$ к высоте зеркала. Исходя из этого можем записать формулу для t рядов зеркала: $H = (n + k) * t + k$

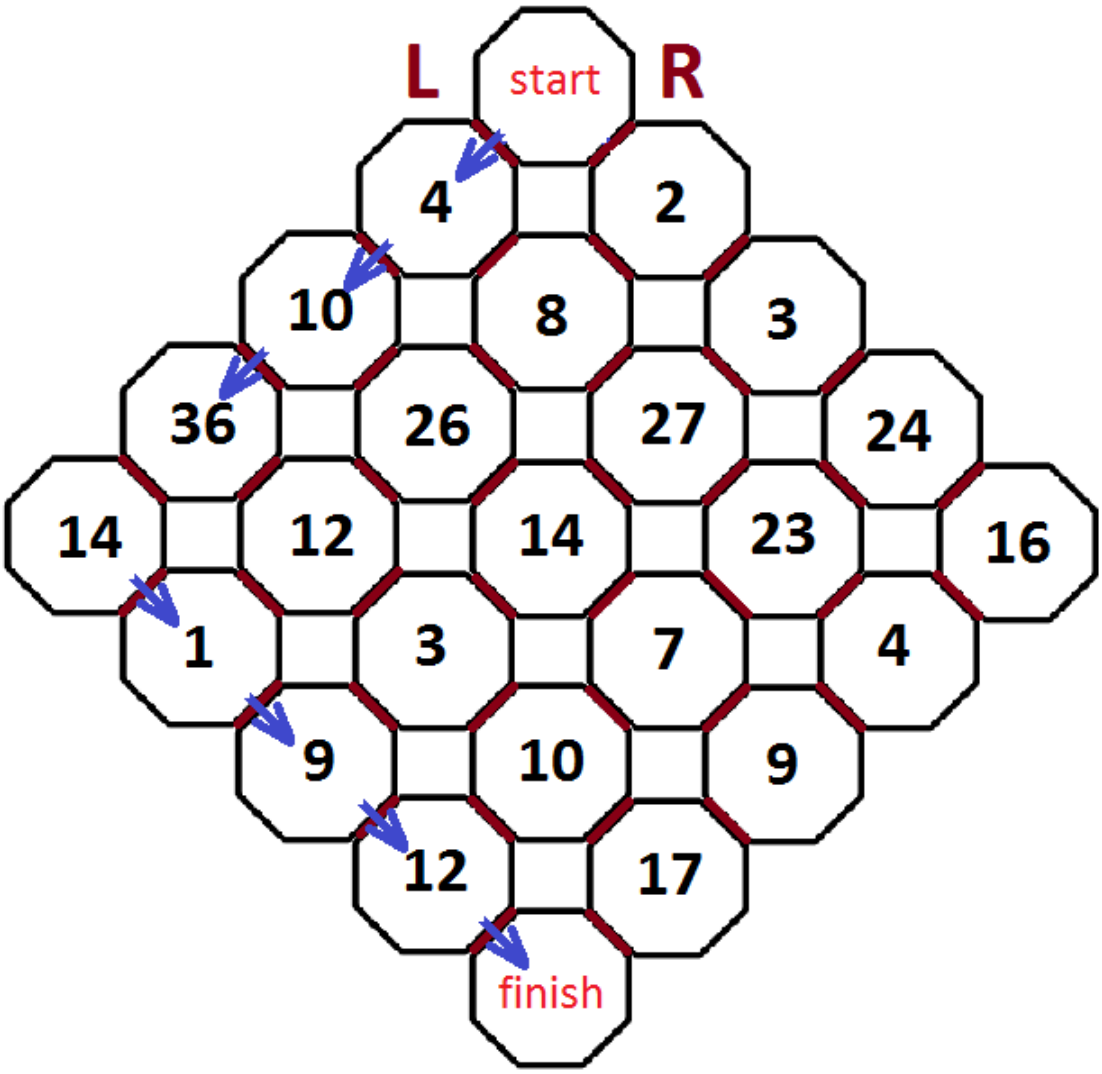
Так как по условию количество рядов всегда чётное, то количество плиток в первых двух рядах $m + m - 1$. Тогда количество плиток, составляющих всю зеркальную поверхность можем посчитать по формуле: $B = (2 * m - 1) * t/2$

Ответы:

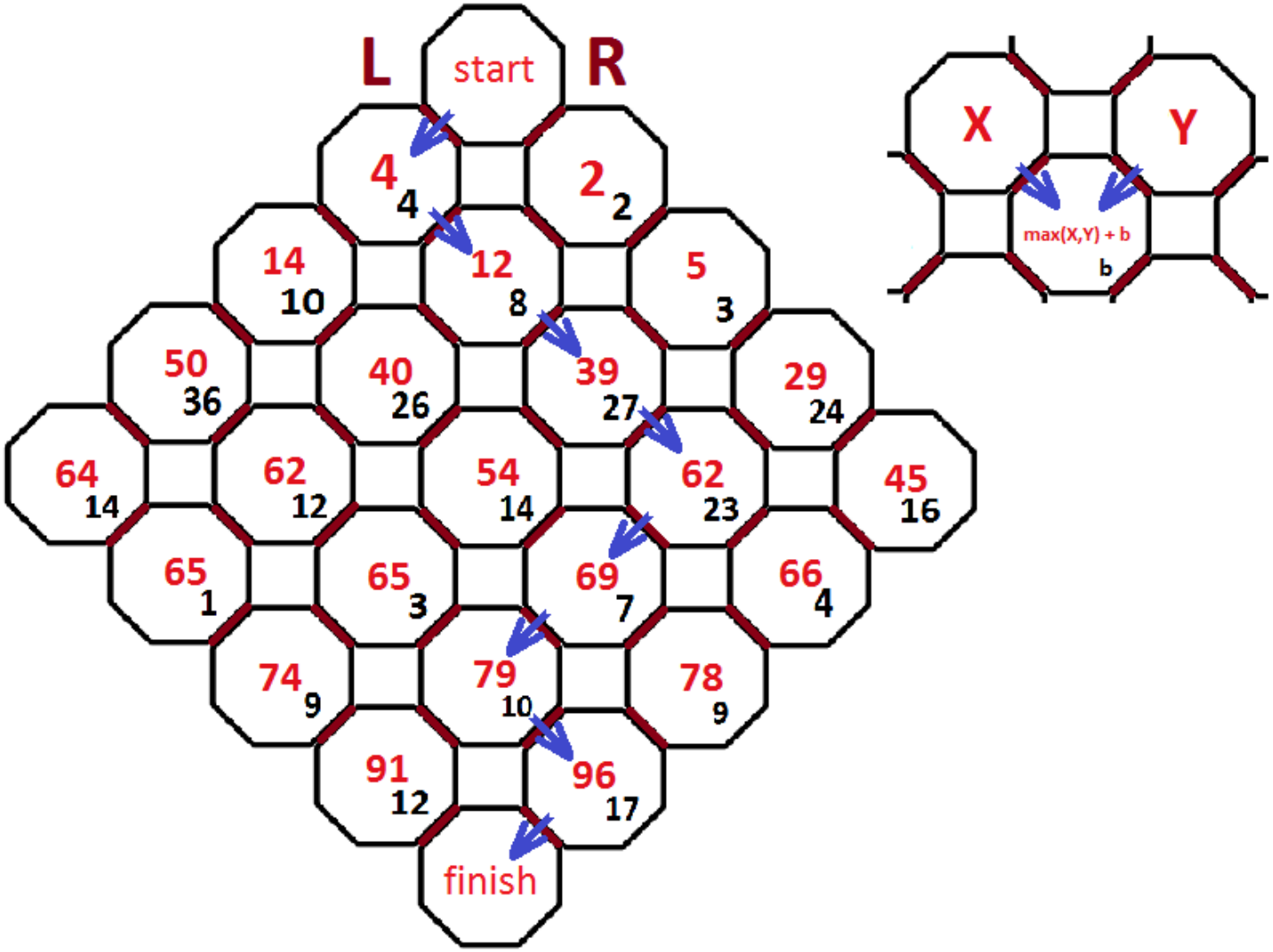
n	k	t	m	H	B
10	5	4	5	65	18
17	7	8	10	199	76
64	16	14	15	1136	203
99	33	20	25	2673	490
49	7	50	49	2807	2425

Задача 2. Лягушонок Пепе

Жадная стратегия – движение в сторону большего количества биткоинов – позволит набрать в данной задаче 50 баллов



Эффективное решение – метод динамического программирования – будем хранить для каждой комнаты **максимальную сумму**, которую можем получить двигаясь от старта до этой комнаты. При пересчете выбираем каждый раз из двух вариантов, соответствующих способам попасть в комнату, максимальный. Для восстановления ответа – пройдем от конца (последней комнаты) к началу в обратном порядке – каждый раз «двигаясь в сторону» максимального значения



Задачу можно решать простым перебором вариантов – в данном случае всего 70 вариантов, но, очевидно, более менее разумных не так много.

Ответ:

96

LRRRLLRL

Задача 3. Водолей

Перед тем, как решать задачу необходимо попробовать получить различный уровень воды в сосудах A и B.

Чтобы получить 6 литров в сосуде B необходимо дважды наполнить сосуд A, а затем перелить его содержимое в сосуд B.

Чтобы получить 4 литра в сосуде B необходимо наполнить сосуд B, а затем перелить его содержимое в пустой сосуд A. В сосуде B останется 4 литра.

Чтобы получить 1 литр в сосуде B необходимо наполнить сосуд B, а затем перелить его содержимое в пустой сосуд A, опустошить A и еще раз перелить воду в пустой сосуд A.

Чтобы получить 2 литра в сосуде A необходимо получить 6 литров в сосуде B, набрать воду в пустой сосуд A и перелить ее в сосуд B. В сосуде A останется 2 литра.

Заметим также, что:

- команда «наполнить A» не изменит состояние сосудов, если сосуд A полный
- команда «опустошить A» не изменит состояние сосудов, если сосуд A пустой
- команда «перелить в A» не изменит состояние сосудов, если сосуд A полный
- команда «перелить в A» не изменит состояние сосудов, если сосуд B пустой

Рассмотрим получение правильной последовательности для случая $A = 2, B = 7$. Для того, чтобы получить заданное количество воды нужно выполнить команды: «наполнить A», «перелить в B», «наполнить A», «перелить в B», «наполнить A», «перелить в B». Расставим полученные команды в заданной последовательности (в скобках укажем объем воды в каждом сосуде, после выполнения команды):

1. наполнить A $(A = 3, B = 0)$
2. перелить в B $(A = 0, B = 3)$
3. опустошить ...
4. наполнить A $(A = 3, B = 3)$
5. перелить в B $(A = 0, B = 6)$
6. опустошить ...
7. перелить в ...
8. опустошить ...
9. наполнить A $(A = 3, B = 6)$
10. перелить в B $(A = 2, B = 7)$

Оставшиеся позиции нужно заполнить так, чтобы количество воды в сосудах не менялось. В результате получаем: $ABAABABAAB$

Для остальных случаев ответ получаем аналогично.

Задача 4. Тайна Карбофоса

Заметим два важных факта:

- нет смысла поворачивать ручку несколько раз: любое нечётное количество поворотов ручки соответствует одному повороту, любое чётное количество поворотов равносильно состоянию, когда ручку не поворачивали
- порядок поворота ручек не важен

Закодируем положение ручек 0 - горизонтальное, 1 - вертикальное. Тогда начальное положение обозначим последовательностью 011010101. Построим таблицу, в которой первая строка соответствует начальному положению, а каждая следующая соответствует повороту одной из ручек (первой, второй, третьей и так далее)

Ручка	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
-	0	1	1	0	1	0	1	0	1
1	1					1			1
2		1	1					1	
3			1			1	1		1
4		1	1	1					
5	1			1	1	1			
6		1				1	1		
7				1	1		1	1	
8			1					1	1
9	1			1					1

Для получения ответа нужно выбрать строки таким образом, чтобы в каждом столбце получилось чётное количество единиц для выбранных строк. Поиск ответа начнем с ручки с номером 5, так как исходно она повернута вертикально и ее поворота можно добиться только одним из двух способов: непосредственным поворотом ручки 5 или поворотом ручки 7. Таким образом, в правильном ответе (если он существует) **обязательно** должна быть или ручка 5 или ручка 7, но не обе эти ручки.

Поворот ручки 5 приводит к положению 111101101. Ручки 5 и 7 нужно вычеркнуть, так как их больше поворачивать нельзя.

Ручка	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
-	1	1	1	1	0	1	1	0	1
1	1					1			1
2		1	1					1	
3			1			1	1		1
4		1	1	1					
6		1				1	1		
8			1					1	1
9	1			1					1

Следующая ручка, которую будем рассматривать - это ручка 1, так как она в настоящий момент повернута и при этом есть только два варианта, чтобы повернуть ее горизонтально: непосредственно повернуть ее или повернуть ручку 9. Попробуем повернуть ручку 1, тогда получим:

Ручка	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
-	0	1	1	1	0	0	1	0	0
2		1	1					1	
3			1			1	1		1
4		1	1	1					
6		1				1	1		
8			1					1	1

Очевидно, что ручку 4 поворачивать обязаны - это единственный вариант. После ее поворота получаем:

Ручка	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
-	0	0	0	0	0	0	1	0	0
2		1	1					1	
3			1			1	1		1
6		1				1	1		
8			1					1	1

Теперь требуется повернуть только ручку 7, это можно сделать либо повернув ручку 3, либо повернув ручку 6, однако, при этом оставшимися в рассмотрении ручками (2 и 8) развернуть все ручки в горизонтальное положение не получится.

Вернемся к шагу, на котором поворачивали ручку 1 и вместо этого повернем ручку 9. Получим:

Ручка	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
-	0	1	1	0	0	1	1	0	0
2		1	1					1	
3			1			1	1		1
4		1	1	1					
6		1				1	1		
8			1					1	1

В этом случае ручку 4, очевидно, также можно исключить, так как ее нельзя поворачивать. Для поворота ручки 2 есть два варианта: повернуть ее непосредственно или повернуть ручку 6. Повернем ручку 2 и получим:

Ручка	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
-	0	0	0	0	0	1	1	1	0
3			1			1	1		1
8			1					1	1

Повернув оставшиеся две ручки с номерами 3 и 8 получим горизонтальное положение для каждой ручки чемодана. Таким образом, получили один из двух вариантов верных ответов.

Варианты верных ответов (порядок ручек не влияет на баллы): 23589 или 23467

Замечания:

- для того, чтобы получить второй верный ответ, необходимо в качестве первой поворачиваемой ручки выбрать ручку с номером 7, а дальше действовать аналогично
- в этой задаче можно получить частичные баллы, если подобрать последовательность, в результате применения которой количество ручек, повернутых вертикально не будет больше 3
- задачу можно решить методом перебора всех вариантов на компьютере с использованием языка программирования, в этом случае потребуется перебрать 512 вариантов
- ускорить процесс подбора нужных строк можно с использованием электронных таблиц MS Excel (функции СУММ и ОСТАТ)

Задача 5. Фермер

Прочитаем данные и будем перебирать каждый килограмм кукурузы от 1 до n в цикле. Если номер очередного килограмма не делится на k, то добавляем к ответу стоимость этого килограмма a.

```
n = int(input())
a = int(input())
k = int(input())

ans = 0
for i in range(1, n+1):
  if i % k != 0:
    ans += a
print(ans)
```

Решение циклом можно ускорить. Для этого на каждом шаге будем рассматривать не по одному килограмму, а по k килограмм. Из каждых таких k килограмм к ответу будем прибавлять $(k-1)$ килограмм. Оставшееся количество килограмм (меньше k) добавим к ответу

```
n = int(input())
a = int(input())
k = int(input())

ans = 0
while n >= k:
    ans += a * (k - 1)
    n -= k

print(ans + a * n)
```

Последнее решение нетрудно превратить в формулу:

```
n = int(input())
a = int(input())
k = int(input())

print( (n // k * (k - 1) + n % k) * a )
```

В более компактном виде формула выглядит следующим образом: $(n - n//k) * a$. Здесь $n//k$ — количество килограмм, которые достанутся Джону бесплатно

Задача 6. Боинг

Для решения задачи необходимо получить номер места в ряду и номер ряда.

Чтобы получить номер ряда можно использовать формулу $(n+3)//6+1$ — считаем что в каждом ряду по 6 мест со смещением 3 от начала.

Для того, чтобы по номеру места получить номер буквы в латинском алфавите можно воспользоваться формулой: $(n+3)\%6$ — используется всего 6 букв со смещением 3 относительно начала алфавита.

Случаи, когда n принимает значения $117, 118, 119$ обработаем отдельно с использованием условного оператора.

```
n = int(input())
if n == 119:
  print("full")
elif n == 118:
  print("21E")
elif n == 117:
  print("21D")
else:
  s = "ABCDEF"
  j = (n + 3) % 6
  print((n + 3) // 6 + 1, s[j], sep="")
```

Задача 7. Интересное подмножество

Для неэффективного решения данной задачи необходимо написать перебор различных вариантов подмножества, например, с использованием рекурсии или битовых масок.

Отдельно отметим случай, в котором размер исходного множества равен 5 — в этом случае для получения решения можно написать 5 циклов для поиска ответа.

```
n = int(input())
ans = 0

if n == 5:
    a = [int(input()) for i in range(5)]

    for i0 in False, True:
      for i1 in False, True:
        for i2 in False, True:
          for i3 in False, True:
            for i4 in False, True:
              b = []
              if i0:
                b.append(a[0])
              if i1:
                b.append(a[1])
              if i2:
                b.append(a[2])
              if i3:
                b.append(a[3])
              if i4:
                b.append(a[4])
              for x in b:
                if x <= len(b):
                  break
              else:
                if len(b) > ans:
                  ans = len(b)
print(ans)
```

Для решения задачи на полный балл необходимо учесть, что все элементы множества попарно различны и подаются на вход в отсортированном по возрастанию виде.

Заметим, что для того, чтобы найти размер наибольшего интересующего нас подмножества, достаточно найти первый (наименьший) элемент исходного множества такой, что количество элементов больших его по значению в исходном множестве, меньше чем значение этого элемента.

```
n = int(input())
a = [int(input()) for i in range(n)]
for i in range(n):
  if a[i] > n - i:
    print(n - i)
    break
else:
  print(0)
```

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).

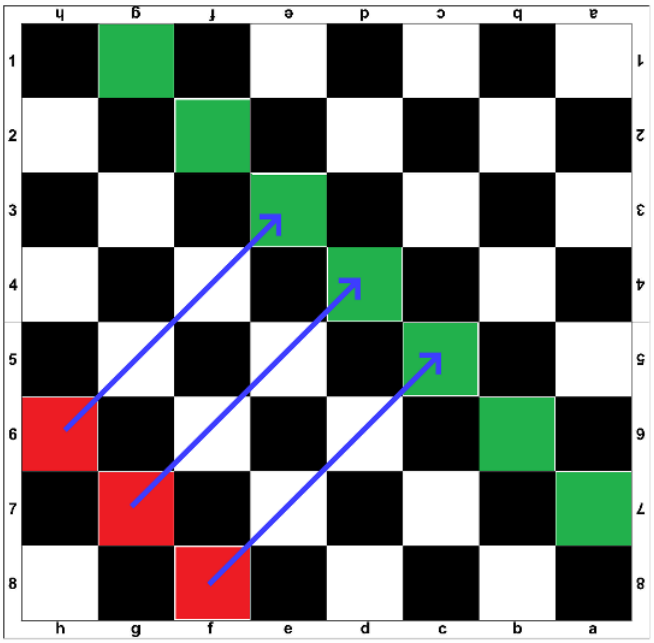
   Для этого на каждом шаге будем делать один ход в ее направлении, уменьшая или увеличивая координаты текущей клетки на 1. Код на языке Python представлен ниже:

```
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.



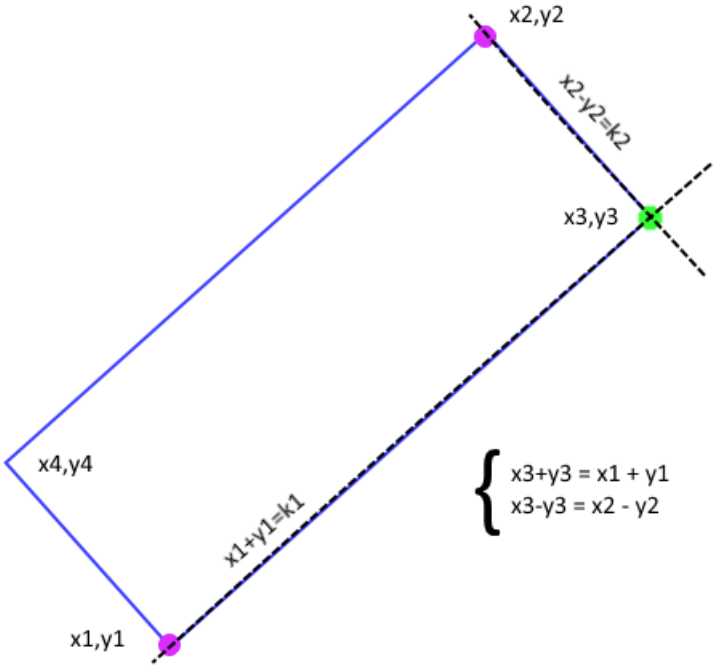
Таким образом, нам достаточно перебором найти клетку шахматного поля, из которой можно попасть ходом слона как в начальную, так и в конечную клетки. Код на языке Python приведен ниже:

```
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.