

Разбор задач

Задача 1. Знакомства

За вечер каждому гостю надо познакомиться со всеми остальными, то есть с $(n - 1)$ гостем. Но с 2 гостями каждый познакомился в хороводе, получается, что осталось познакомиться с $(n - 3)$ гостями. Если n умножить на $(n - 3)$, то получится удвоенное количество знакомств, потому что каждое знакомство посчитано дважды: знакомство первого человека пары со вторым и второго с первым. Тогда для определения количества знакомств надо разделить это число на 2.

Получаем формулу: $n * (n - 3) / 2$.

Задача 2. Легендарные тренировки

Запишем для каждого игрока набор номеров команд, в составе которых он играл. 1 будет означать первую команду, а 2 - вторую.

1 - 1111
2 - 1112
3 - 1121
4 - 1122
5 - 1211
6 - 1212
7 - 1221
8 - 1222
9 - 2111
10 - 2112
11 - 2121
12 - 2122
13 - 2211
14 - 2212
15 - 2221

При таком распределении каждый игрок сыграл против каждого, потому что если бы два игрока всё время играли в одной команде, то строки распределения у них были бы абсолютно одинаковыми. Так как все строки распределения разные, любая пара игроков отличается командой хотя бы в одной тренировке.

Докажем, что нельзя добиться такого результата менее чем за четыре тренировки. Если бы тренировок было три, то количество различных распределений чисел 1 и 2 на три тренировки составило бы $2^3 = 8$, и их не хватило бы на 15 игроков.

Для приведённого выше распределения в ответ нужно указать такие составы первой команды:

1 2 3 4 5 6 7 8
1 2 3 4 9 10 11 12
1 2 5 6 9 10 13 14
1 3 5 7 9 11 13 15

Есть и другие варианты распределения игроков на команды в четырёх тренировках.

Задача 3. Рыцари и лжецы

Одно из возможных решений:

0	0	1	0	0
1	0	0	0	1
0	0	1	0	0
1	0	0	0	1
0	0	1	0	0

В такой расстановке вокруг каждой единицы стоят только нули, и возле каждого нуля есть хотя бы одна единица. При этом расставлено 7 рыцарей.

С помощью полного перебора вариантов в компьютерной программе можно доказать, что не выйдет расставить на таком поле меньше 7 рыцарей, удовлетворяя условиям задачи.

Существуют и другие расстановки 7 рыцарей.

Задача 4. Игра

Для каждой клетки, начиная с нижнего уровня, вычислим максимальное количество монет, которое можно набрать, добравшись до неё:

10	50	50	48
9	50	45	45
8	35	35	45
7	35		21
6	35	21	21
5		15	21
4	12	12	14
3	12	2	6
2	2		0
1	2	0	0

По этой таблице видно, что максимальное количество набранных монет равно 50. Восстановим путь, двигаясь по которому, можно набрать эти 50 монет:

10	50	50	48
9	50	45	45
8	35	35	45
7	35		21
6	35	21	21
5		15	21
4	12	12	14
3	12	2	6
2	2		0
1	2	0	0

Получаем программу-решение:

```
#>##< <####>
```

Задача 5. Лёша-путешественник

В этой задаче нужно заметить, что каждое из двух условий (что перед Лёшей находится не менее A вагонов, а за ним – не более B вагонов) задаёт суффикс вагонов, в которых может находиться Лёша. Из первого условия следует, что Лёша должен находиться в $N - A$ последних вагонах, а второе условие говорит, что Лёша должен находиться в $B + 1$ последних вагонах. Таким образом, ответом является $\min(N - A, B + 1)$.

```
n = int(input())
a = int(input())
b = int(input())
ans = min(n - a, b + 1)
print(ans)
```

Задача 6. Ход слона

Рассмотрим решение, работающее при $R = C$. В таком случае слон обязан находиться на диагонали, соединяющей левый нижний угол с правым верхним. Тогда слон обязательно бьёт все клетки данной диагонали за исключением той клетки, на которой стоит он сам.

Также слон бьёт клетки на ещё одной диагонали, которая параллельна главной (идущей из левого верхнего угла в правый нижний). Количество клеток, которые он бьёт, тем больше, чем ближе слон к центру доски. Рассмотрим количество клеток, которые слон бьёт на этой диагонали в зависимости от его координат, при $N = 6$:

- 0
- 2
- 4
- 4
- 2
- 0

Видно, что по мере приближения к центру количество атакуемых клеток увеличивается с шагом 2. Получаем следующий вариант решения:

```
n = int(input())
r = int(input())
c = int(input())
ans = n - 1
if r <= (n + 1) // 2:
    ans += r + c - 2
else:
    ans += 2 * n - r - c
print(ans)
```

Решение, работающее при $N \leq 100$, построить несколько проще, чем предыдущее. Достаточно воспользоваться весьма простым свойством — если две клетки расположены на одной диагонали, то модуль разности между номерами их строк равен модулю разности между номерами их столбцов. Остаётся только перебрать все клетки и посчитать, для какого количества клеток справедливо данное равенство с клеткой, где стоит слон, не забыв учесть, что для вышеупомянутой клетки данное равенство тоже выполнится.

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

```
ans = 0
for row in range(1, n + 1):
    for col in range(1, n + 1):
        if abs(row - r) == abs(col - c):
            ans += 1
print(ans - 1)
```

Для того чтобы получить полное решение, необходимо по координатам слона определять длины диагоналей, на которых он находится.

Рассмотрим диагонали, параллельные главной. Длина главной диагонали равна N , соседних ей диагоналей – $N - 1$, соседних соседним – $N - 2$ и так далее. Следовательно, длина таких диагоналей будет равна разнице N и расстояния до главной диагонали, что приводит нас к формуле $N - \text{abs}(R + C - 1 - N)$, где $R + C - 1$ номер диагонали.

Подобные размышления применимы и к диагоналям, которые параллельны побочной диагонали, с той лишь разницей, что номер диагонали считается по формуле $N - R + C$, а значит, итоговая формула для вычисления длины диагонали будет иметь вид $N - \text{abs}(C - R)$.

Ответом на задачу является сумма длин диагоналей, на которых лежит клетка со слоном, уменьшенная на два, чтобы учесть занятую слоном клетку. Итоговое решение на языке Python получится таким:

```
n = int(input())
r = int(input())
c = int(input())
len_diag_main = n - abs(r + c - 1 - n)
len_diag_sub = n - abs(c - r)
print(len_diag_main + len_diag_sub - 2)
```

Задача 7. Озеленение

Разобьём решение на 2 части:

1) Пусть мы зафиксировали какие-то стороны участка и их ориентацию, обозначим их как A и B . Заметим, что по стороне длины N мы можем разместить не более $N//A$ участков, а по стороне длины B – не более $M//B$. Тогда всего получится разместить не более $(N//A) * (M//B)$ участков. Заметим, что если поменять A и B местами, то формула примет вид $(N//B) * (M//A)$. Таким образом, по заданным A и B мы научились за $O(1)$ искать наибольшее количество участков, которые удастся разместить на прямоугольнике $N \times M$.

2) Теперь научимся быстро перебирать A и B . Вспомним, что $A * B = S$, заметим, что можно перебирать только A , а $B = S/A$. Теперь воспользуемся следующим фактом: если $A * B = S$, то $\min(A, B) \leq \sqrt{S}$. Достаточно перебрать только числа, которые меньше или равны \sqrt{S} , а второй делитель будет находиться однозначно.

Таким образом, мы получили решение за $O(\sqrt{S})$

```
n = int(input())
m = int(input())
s = int(input())
ans = 0
d = 1
while d * d <= s:
    if s % d == 0:
        a = d
        b = s // d
        ans = max(ans, (n // a) * (m // b), (n // b) * (m // a))
    d += 1
print(ans)
```