Разбор задач

Задача 1. Костяные войны

Частичное решение:

Переберем все возможные a и b, не превышающие P, и проверим выполнение равенства $2(a+b)=P$.

```
p = int(input())
count = 0
for a in range(1, p + 1):
    for b in range(1, p + 1):
        if 2 * (a + b) == p:
            count += 1
print(count)
```

Полное решение:

Очевидно, что если периметр нечётен, то способов нет ни одного, и нужно вывести ноль.

В чётном случае заметим, что один из них, пускай Марш, может пожертвовать отрезки длиной от 1 до $P/2-1$, а отрезки Копа определятся однозначно. Всего $P/2-1$ способ.

```
n = int(input())
if n % 2 == 1:
    print(0)
else:
    print(n // 2 - 1)
```

Задача 2. Потерявшееся число

Частичное решение:

Будем перебирать все числа, делящиеся на 6, получать их половину и треть и проверять, равна ли какая-нибудь пара чисел из этой тройки паре чисел, данной в условии. Если да — ответом будет третье число из тройки, если нет — переходим к следующему числу, делящемуся на 6, и проверяем его.

Полное решение:

Первый случай: остались карточки, на которых изначально были записаны исходное число и его половина, потерялась карточка с третью числа.

Признаком этого случая является следующее равенство: 2 * a = b. Значит, третье число можно вычислить по формуле b // 3.

Второй случай: остались карточки, на которых изначально были записаны исходное число и его треть, потерялась карточка с половиной числа.

Признаком этого случая является следующее равенство: 3 * a = b. Значит, третье число можно вычислить по формуле b // 2.

Третий случай: остались карточки, на которых изначально были записаны половина исходного число и его треть, потерялась карточка с самим числом.

Признаком этого случая является следующее равенство: 3 * a = 2 * b. Значит, третье число можно вычислить по формуле 3 * a.

```
a = int(input())
b = int(input())
if a * 2 == b:
    print(b // 3)
elif a * 3 == b:
```

```
    print(b // 2)
else:
    print(2 * b)
```

Задача 3. Два грузчика

Частные решения:

Первая подзадача (любую коробку может унести любой грузчик):

В этом случае каждой ходкой переносится по 2 коробки, поэтому...

Первый случай: если n — четное: $ans = n/2$.

Второй случай: если n — нечетное: $ans = n/2 + 1$ (последнюю коробку несет кто-то один).

В обеих случаях ответом будет $n/2$, округленное вверх до целого числа или $ans = \lfloor (n+1)/2 \rfloor$, где операция $\lfloor . \rfloor$ — целая часть от результата деления.

```
b = int(input())
n = int(input())
ans = (n + 1) // 2
print(ans)
```

Вторая подзадача (нет коробок, которые нужно тащить вдвоем):

Пусть x — количество коробок, которые может перенести Шурик, y — количество коробок, которые может перенести Федя, но не может Шурик (считывая вес очередной коробки будем увеличивать на 1 соответствующую переменную).

Первый случай: если $x \geqslant y$. В этом случае Федя перетаскает все свои коробки не раньше Шурика и общее количество ходок получится таким же, как и в первой подзадаче: ответом будет $n/2$ округленное вверх до целого числа или $ans = \lfloor (n+1)/2 \rfloor$.

Второй случай: если $y > x$. В этом случае Федя перетаскает все свои коробки позже Шурика и несколько последних ходок Шурик будет просто идти рядом с Федей с пустыми руками: $ans = y$.

```
b = int(input())
n = int(input())
L = list()
for i in range(n):
    L.append(int(input()))

x, y = 0, 0
for i in L:
    if i <= a:
        x += 1
    else:
        y += 1

ans = 0
if x <= y:
    ans += y
else:
    ans += (x + y + 1) // 2
print(ans)
```

Полное решение:

Пусть x — количество коробок, которые может перенести Шурик, y — количество коробок, которые может перенести Федя, но не может Шурик, z — количество коробок, которые они могут перенести только вдвоем.

Тогда ответ точно равен z (столько коробок они несут вдвоём) плюс...

Первый случай: если $x \leqslant y$, то к ответу добавим y (переходов Феди с грузом больше, чем у Шурика).

Второй случай: иначе добавим $(x+y)/2$ с округлением вверх (Шурик не ходят без груза, кроме, возможно, последнего перехода, если осталась одна коробка на двоих).

```
a = int(input())
b = int(input())
n = int(input())
x, y, z = 0, 0, 0
for i in range(n):
    p = int(input())
    if p <= a:
        x += 1
    elif p <= b:
        y += 1
    else:
        z += 1
ans = z
if x <= y:
    ans += y
else:
    ans += (x + y + 1) // 2
print(ans)
```

Задача 4. Любовь и двери

Далее под «правильным набором дверей» будем понимать такой набор, при котором изменять их уже не нужно, то есть в начале набора стоят двери одного типа, а в конце двери другого типа, например набор 2, 2, 2, 2, 1, 1, 1 будет правильным. Такую проверку можно сделать за одно прохождение по набору путем сравнения рядом стоящих дверей. Если не более одной пары рядом стоящих дверей различны, то такой набор является правильным.

Решение для тестов первой группы, в которых не больше трех золотых дверей и длина которых не больше 10. Понятно, что в этом случае ответ не превосходит 3. Попробуем перебрать все варианты изменения 0, 1 или 2 дверей и для каждого из них проверим, что полученный после текущего изменения набор дверей является правильным. После каждого изменения и проверки на правильность отменяем все изменения и проверяем следующий вариант. При этом вариант с 0 изменений (то есть без изменений) проверяется сразу, вариант с одним изменением проверяется одним циклом, а вариант с двумя изменениями проверяется двойным вложенным циклом. Максимальная сложность будет кубической с учетом проверки каждого варианта на правильность.

Решение для тестов второй группы, в которых длина набора не превосходит 100. В этом случае можно реализовать алгоритмы с тройным вложенным циклом. Например такой: переберем место встречи героев i (до позиции i невключительно мы хотим чтобы были двери одного типа, а после этой позиции включительно — двери другого типа). Далее для выбранной границы бежим по всем дверям и каждую дверь, которая не соответствует зафиксированному выше порядку дверей для позиции i изменяем, при этом увеличивая счетчик. Непосредственно после каждого изменения проверяем является ли набор после этого изменения правильным и если да — то сравниваем полученное до этого момента количество изменений с минимальным. Не забудем проверить два варианта расположения дверей (сначала двери типа 1, потом двери типа 2 и наоборот).

Для прохождения тестов третьей группы, в которых длина набора не больше 1000 нужно избавиться от одного вложенного цикла, то есть написать квадратичное решение. Можно заметить, что в предыдущем решении не обязательно проверять на правильность набор дверей после каждого изменения, достаточно это делать только после полного приведения дверей к порядку, зафиксиро-

ванному при помощи позиции встречи i. Другой вариант с такой же сложностью (наиболее близкий к основному решению всей задачи) такой: мы для позиции встречи i и заданного этой позицией расположения дверей просто переберём все двери и подсчитаем количество дверей, которые нужно изменить слева, и количество дверей, которые нужно изменить справа, чтобы получить заданную расстановку дверей. Среди всех таких вариантов изменения найдем минимальный.

Для получения полного решения нужно написать линейное решение. После чтения последовательности подсчитаем сколько в ней цифр 1 и сколько цифр 2. Далее за один проход по набору получим ответ следующим образом: для каждой позиции i от 0 до длины набора минус один мы храним $left_1$ — сколько цифр 1 и $left_2$ — сколько цифр 2 было встречено на отрезке от начала строки до позиции $i-1$ включительно (для $i=0$ эти величины полагаем равными 0). Зная общее количество цифр по отдельности и величины $left_1$ и $left_2$, можно вычислить сколько раз встречается каждая цифра на отрезке от i до конца строки ($right_1$ и $right_2$ соответственно). Тогда для позиции i вычислим минимум из $left_1 + right_2$ и $left_2 + right_1$ — столько минимум нужно сделать замен, чтобы именно в этой позиции произошла встреча. Осталось выбрать самую маленькую по этому показателю позицию и вывести минимум для неё.

```
n = int(input())
a = [int(input()) for i in range(n)]
left_1 = 0
left_2 = 0
right_1 = a.count(1)
right_2 = a.count(2)
ans = min(right_1, right_2)
for c in a:
    if c == 1:
        right_1 -= 1
        left_1 += 1
    else:
        right_2 -= 1
        left_2 += 1
    ans = min(ans, left_1 + right_2, left_2 + right_1)
print(ans)
```

При написании и тестировании программы полезно проверить особые частные случаи.

Первый случай — когда двери изначально уже расположены так, что их можно открыть без изменений (тогда ответом будет число 0). Например, задана последовательность 2, 1, 1, 1. Очевидно, что если выдать тому кто у первой двери серебряный ключ, а тому кто у последней двери золотой, то можно открыть сразу все двери без их изменения. Важным подслучаем здесь является вариант, когда все двери уже изначально имеют один тип, например 2, 2, 2, 2, 2 или 1, 1, 1. Эти примеры тоже обязательно нужно протестировать.

Второй частный случай связан с вариантом, когда оптимально изменять только двери одного типа, например серебряные на золотые, и окончательный набор дверей так же является последовательностью дверей одного типа (в данном случае золотых). Например, последовательность 1, 1, 1, 1, 2, 1, 1, 1, 1. Для неё лучшим вариантом будет изменить за одну минуту дверь номер 5.

Задача 5. Долгое вычитание, Карл!

Частные решения.

Первая подзадача.

Если число n — однозначное, то ответ равен самому числу. Если число n — двузначное четное, то потребуется $\lfloor (n-8)/2 \rfloor$ операций (через $\lfloor \cdot \rfloor$ обозначается целая часть от деления), чтобы число стало равно 8. Итого `ans = (n - 8) // 2 + 8`. Если число n — двузначное нечетное, то потребуется $\lfloor (n-9)/2 \rfloor$ операций, чтобы число стало равно 9. Итого `ans = (n - 9) // 2 + 9`.

```
n = int(input())
if n % 2:
    ans = (n - 9) // 2 + 9
else:
    ans = (n - 8) // 2 + 8
if n < 10:
    ans = n
print(ans)
```

Вторая подзадача.

Моделирование процесса вычитания «в чистом виде»: пока число не обратится в 0, вычитать из него его длину, подсчитывая количество операций.

```
n = int(input())
ans = 0
while n:
    ans += 1
    n -= len(str(n))
print(ans)
```

Полное решение.

Разберем конкретный пример. Пусть $n = 123456$.

Найдем первое пятизначное число, встретившееся нам в процессе уменьшения n. Очевидно, оно имеет такой же остаток от деления на 6, что и n. Возьмем число 99999 и будем уменьшать его на 1, пока остатки не равны. Получим новое $n = 99996$. При этом мы использовали $\lfloor(123456 - 99996)/6\rfloor = 3910$ операций.

Далее аналогично. Найдем первое четырехзначное число, встретившееся нам в процессе уменьшения n. Очевидно, оно имеет такой же остаток от деления на 5, что и n. Возьмем число 9999 и будем уменьшать его на 1, пока остатки не равны. Получим новое $n = 9996$. При этом мы использовали $\lfloor(99996 - 9996)/5\rfloor = 18000$ операций (всего 21910).

Найдем первое трехзначное число, встретившееся нам в процессе уменьшения n. Очевидно, оно имеет такой же остаток от деления на 4, что и n. Возьмем число 999 и будем уменьшать его на 1, пока остатки не равны. Получим новое $n = 996$. При этом мы использовали $\lfloor(9996 - 996)/4\rfloor = 2250$ операций (всего 24160).

Найдем первое двузначное число, встретившееся нам в процессе уменьшения n. Очевидно, оно имеет такой же остаток от деления на 3, что и n. Возьмем число 99 и будем уменьшать его на 1, пока остатки не равны. Получим новое $n = 99$. При этом мы использовали $\lfloor(996 - 99)/3\rfloor = 299$ операций (всего 24459).

Наконец, найдем первое однозначное число, встретившееся нам в процессе уменьшения n. Очевидно, оно имеет такой же остаток от деления на 2, что и n. Возьмем число 9 и будем уменьшать его на 1, пока остатки не равны. Получим новое $n = 9$. При этом мы использовали $\lfloor(99 - 9)/2\rfloor = 45$ операций (всего 24504).

Не забудем, что n нужно уменьшить до 0, поэтому добавим к ответу еще 9. Итого 24513.

Запрограммируем этот процесс: будем последовательно находить наибольшие числа длины меньше, чем исходное на 1, которое встретится нам в процессе вычитания (и будет последним, полученным при вычитании длины исходного числа), пока исходное число не уменьшится до однозначного.

```
n = int(input())
ans = 0
while n > 9:
    new_n = int('9' * (len(str(n)) - 1))
    while new_n % len(str(n)) != n % len(str(n)):
```

```
        new_n -= 1
    ans += (n - new_n) // len(str(n))
    n = new_n
ans += n
print(ans)
```