Разбор задач

Задача 1. Пара-тройка конфет

Заметим, что упаковки конфет могут быть двух типов: с двумя шоколадными + одной карамельной или одной шоколадной + двумя карамельными. Также заметим, что ответ не превосходит n, так как в каждой упаковке есть хотя бы одна конфета каждого вида.

При $n \leqslant 10^3$ можно перебрать количество купленных коробок и убедиться, что при любом распределении конфет в купленных упаковках найдётся не менее n конфет одного вида.

Пусть k — количество купленных коробок, из них k_1 — первого типа (две шоколадные и одна карамельная), тогда $k_2 = k - k_1$ — число коробок второго типа (одна шоколадная и две карамельные). Конфет одного вида будет $\max(2k_1 + k_2, k_1 + 2k_2)$. Для выбранного k переберём значения k_1 такие, что $0 \leqslant k_1 \leqslant k$ и если для всех k_1 верно, что $\max(2k_1 + k_2, k_1 + 2k_2) \geqslant n$, то данное значение k подходит. Минимальное подходящее k является ответом.

Такое решение имеет сложность $O(n^2)$ и набирает 25 баллов. Пример такого решения.

```
n = int(input())

k = 1
ok = False
while not ok:
    ok = True
    for k1 in range(k + 1):
        k2 = k - k1
        if max(2 * k1 + k2, k1 + 2 * k2) < n:
            ok = False
    k += 1
print(k - 1)
```

При $n \leqslant 10^6$ всё ещё получится перебирать значения k, но необходимо избавиться от вложенного цикла, перебирающего возможное распределение упаковок. Заметим, что в худшем случае (чтобы минимизировать количество конфет одного вида), значение $k_1 \approx k_2$, то есть конфет каждого вида окажется примерно поровну.

Заметим, что в двух упаковках точно найдётся три конфеты одного вида. Если мы покупаем чётное число k упаковок, то среди них точно будет $3k/2$ конфет одного вида. Если k нечетное, то количество конфет одного вида будет не меньше, чем $\lfloor 3k/2 \rfloor + 2$. Переберём значения k, пока не найдём минимальное подходящее.

Такое решение имеет сложность $O(n)$ и набирает 50 баллов. Пример такого решения.

```
n = int(input())
k = 0
while k // 2 * 3 + (k % 2) * 2 < n:
    k += 1
print(k)
```

Полное решение имеет сложность $O(1)$. Если n делится на 3, то нужно купить $2n/3$ упаковок (в каждых двух упаковках есть три конфеты одного вида). Если же купить на одну упаковку больше, то в лишней упаковке окажутся 2 одинаковые конфеты. Поэтому для тех значений n, которые не делятся на 3, нужно будет купить ещё одну дополнительную упаковку.

Пример такого решения.

```
n = int(input())
if n % 3 == 0:
    print(n // 3 * 2)
else:
    print(n // 3 * 2 + 1)
```


Сайт 100ballnik.com. Варианты для подготовки.

Задача 2. Речной бой

Будем стрелять последовательно так, чтобы не пропустить корабль. Для этого нужно стрелять, оставляя между выстрелами $k-1$ пустую клетку, то есть в клетки с номерами k, $2k$, $3k$ и т.д., до тех пор, пока не получим результат «ранен», либо такие клетки не кончатся. Мы сделаем не более, чем $\lfloor n/k \rfloor$ выстрелов (целочисленное частное от деления n на k). Получив на каком-то выстреле ответ «ранен», начнём стрелять в соседние клетки в одну сторону, а при получении результата «мимо» — в другую сторону. Тем самым мы сделаем не более одного дополнительного промаха, то есть для попадания в $k-1$ клетку корабля понадобятся не более k выстрелов, и общее число выстрелов будет равно $\lfloor n/k \rfloor + k$.

Но если n делится на k, то последний выстрел из первой последовательности придётся в последнюю клетку поля, и тогда с одной стороны от этой клетки больше не останется клеток. В этом случае достаточно только $k-1$ дополнительных выстрелов, то есть при n, делящемся на k, ответ равен $\lfloor n/k \rfloor + k - 1$.

Используя целочисленное деление, оба этих случая можно записать одной формулой.

```
n = int(input())
k = int(input())
print((n - 1) // k + k)
```

Задача 3. Красивый шарф

В решении на 40 баллов можно явно построить требуемую таблицу. На самом деле не требуется сохранять всю таблицу в массиве, достаточно перебрать номера строк и столбцов. Клетка (i, j) будет иметь цвет $(i + j) \bmod k + 1$, если нумеровать строки и столбцы с нуля. Вычислим цвет клетки и если он равен c, увеличим ответ на 1. Такое решение имеет сложность $O(nm)$.

```
n = int(input())
m = int(input())
k = int(input())
c = int(input())

ans = 0
for i in range(n):
    for j in range(m):
        if (i + j) % k +1 == c:
            ans += 1
print(ans)
```

В решении на 60 баллов нужно убрать вложенные циклы и перебирать только строки или только столбцы. Вычислим для первой строки цвет первой клетки в столбце (это значение запишем в переменную first) и цвет последней клетки (это значение запишем в переменную last), а также количество клеток цвета c в строке, которое будем хранить в переменной count. Значения first и last при переходе к следующей строке увеличиваются на 1 с учётом зацикливания при достижении значения k. Также будем обновлять значение count в зависимости от first и last — если в какой-то строке значение first было равно c, то при переходе к следующей строке этот цвет «сдвинется» влево, и count уменьшится на 1. Аналогично, count увеличивается на 1, когда значение last становится равно c.

Такое решение имеет сложность $O(n + m)$. Его удобнее реализовать, если нумеровать цвета с нуля, тогда обновление переменных first и last при переходе к следующей строке будет выполняться присваиваниями `first = (first + 1) % k`, `last = (last + 1) % k`.

```
n = int(input())
m = int(input())
k = int(input())
c = int(input()) - 1
```

```
answer = 0
first = 0
last = (m - 1) % k
count = 0
for j in range(m):
    if j % k == c:
        count += 1
for i in range(n):
    answer += count
    if first == c:
        count -= 1
    first = (first + 1) % k
    last = (last + 1) % k
    if last == c:
        count += 1
print(answer)
```

Полное решение имеет сложность $O(1)$. Покрашенную в цвет c клетку станем просто называть *покрашенной*, а остальные клетки — *непокрашенными*. Нам необходимо найти количество покрашенных клеток. Заметим, что среди k подряд идущих клеток в одной строке или в одном столбце будет ровно одна покрашенная. Это позволяет отбросить целиком любые k строк: в одной группе из k строк окажется ровно m покрашенных клеток. Таких групп $\lfloor n/k \rfloor$, после чего мы сможем заменить n на остаток от деления n на k. Сделаем то же самое со столбцами: каждая группа из k столбцов будет иметь n покрашенных клеток.

Теперь нам нужно посчитать ответ только для прямоугольника $n \times m$, стороны которого меньше k. Пройдём по строкам и определим количество покрашенных клеток в каждой строке. Оно может быть равно 0 или 1, поэтому просто посчитаем количество строк, в которых такая клетка есть. Например, если в квадрате такая клетка была в первой строке (что возможно в случае $c \leqslant m$), то таких строк будет $\min(c, n)$ — первоначально эта клетка была в столбце c, а затем начнёт сдвигаться влево, пока не дойдёт до первого столбца или пока не кончатся строки. Если же $c > m$, то посчитаем, когда эта клетка «войдёт» в последний столбец, для этого нужно будет пропустить $c - m$ столбцов.

Ещё нужно учесть случай, когда покрашенная клетка сдвинется налево и покинет строку квадрата, а потом снова «войдёт» с правого столбца.

Пример такого решения.

```
n = int(input())
m = int(input())
k = int(input())
c = int(input())

ans = 0
ans += (n // k) * m
n %= k
ans += (m // k) * n
m %= k

if c > m:
    n -= c - m
    c = m
if n > 0:
    ans += min(c, n)
    c += k
    if c > m:
```

```
        n -= c - m
        c = m
        if n > 0:
            ans += min(c, n)
print(ans)
```

Задача 4. Гостиница для жирафов

Пусть имеется набор из k подушек. Всего существует $2^k - 1$ всевозможных подмножеств этого набора, не считая пустого. Поэтому различных вариантов толщины набора будет не более $2^k - 1$. Поэтому, если некоторый набор из k подушек позволит сформировать все стопки толщиной от 1 до n, то

$$2^k - 1 \geqslant n,$$

откуда

$$k \geqslant \lceil \log_2(n+1) \rceil,$$

где $\lceil \cdot \rceil$ — операция округления числа «вверх».

Предъявить подходящий набор, содержащий в точности $p = \lceil \log_2(n+1) \rceil$ подушек, несложно: для этого достаточно рассмотреть степени двойки 2^i для всех i от 0 до $p-1$. Представление всех чисел от 1 до n в виде сумм таких чисел соответствует двоичному представлению этих чисел, поэтому такой набор является достаточным для получения всех нужных сумм и минимальным по количеству используемых подушек. Однако суммарная толщина всех подушек может оказаться не минимально возможной: сумма чисел в этом наборе равна $2^p - 1$, что может оказаться больше n, поэтому в таком наборе присутствует некоторая избыточность с точки зрения величины получаемых сумм.

Важно, что нам не нужно получать суммы, большие n. Поэтому давайте в рассматриваемом наборе заменим подушку самой большой толщины 2^{p-1} на подушку толщины

$$W = n - (2^{p-1} - 1),$$

то есть W — разница между n и суммарной толщиной всех остальных подушек в наборе. Нетрудно понять, что такой набор позволяет представить все суммы от 1 до n: действительно, суммы от 1 до $(2^{p-1} - 1)$ можно получить без использования подушки толщины W, а все большие суммы вплоть до n — прибавляя подушку толщины W к некоторой меньшей сумме. Этот набор, как показано ранее, содержит минимальное достаточное количество подушек. Кроме того, суммарная толщина всех подушек в наборе равна n — меньше суммарная толщина в подходящем наборе быть не может, потому что иначе таким набором нельзя получить толщину n. Таким образом, мы получили полностью удовлетворяющий условию задачи набор из p подушек:

$$1,\ 2,\ 4,\ 8, ..., 2^{p-2}, W.$$

Для реализации этого решения будем просто брать последовательные степени двойки, уменьшая значение n и добавляя к ответу в конце оставшееся значение n, если оно оказалось ненулевым.

```
n = int(input())
last = 1
while last <= n:
    print(last)
    n -= last
    last *= 2
if n > 0:
    print(n)
```

Это не единственный возможный ответ, что показывает, в частности, пример для $n = 9$: подходящими наборами являются $\{1, 2, 2, 4\}$, $\{1, 2, 3, 3\}$, $\{1, 1, 3, 4\}$ и $\{1, 1, 2, 5\}$.

Задача 5. Сломанный индикатор

Для начала формализуем условие задачи. Некий индикатор состоит из 9 сегментов, каждый из которых либо гарантированно работает (зажигается какой-то цифрой из данного набора), либо имеет неизвестное состояние (не зажигается ни одной из данных цифр). Задача заключается в нахождении всех цифр, которые состоят только из гарантированно рабочих индикаторов.

Пронумеруем сегменты индикатора сверху вниз, слева направо, то есть самый верхний сегмент будет иметь номер 1, а самый нижний — 9. Для каждой возможной цифры перечислим сегменты, из которой она состоит. Например, цифра 0 состоит из сегментов 1, 2, 4, 6, 8, 9.

Если какая-то цифра могла быть отображена на экране, то соответствующие ей сегменты заведомо исправны. Если несколько цифр могли быть отображены на экране, то исправны сегменты, которые входят в хотя бы одну данную цифру. То есть нужно сделать объединение множеств сегментов, соответствующих данным цифрам. Это удобно реализовать при помощи структуры данных «множество» в языке Python, но можно использовать и массивы из нулей и единиц (где единица будет соответствовать исправному сегменту индикатора) или битовые операции, но тогда реализация станет сложнее.

Для проверки, что какую-то цифру гарантированно удастся показать на индикаторе, множество сегментов этой цифры должно быть подмножеством заведомо исправных сегментов. Для этого в Python есть операция <= для множеств.

Приведём пример решения на языке Python. Здесь segments — это список множеств, соответствующих цифрам, то есть segments[i] — это список сегментов, входящих в цифру i.

После нахождения объединения множеств сегментов переберём все цифры от 0 до 9 и выведем подходящие из них.

```
segments = [
{1, 2, 4, 6, 8, 9},
{3, 4, 8},
{1, 4, 7, 9},
{1, 3, 5, 7},
{2, 4, 5, 8},
{1, 2, 5, 8, 9},
{3, 5, 6, 8, 9},
{1, 3, 6},
{1, 2, 4, 5, 6, 8, 9},
{1, 2, 4, 5, 7}]

res = set()
n = int(input())
for i in range(n):
    d = int(input())
    res |= segments[d]

for d in range(0, 10):
    if segments[d] <= res:
        print(d)
```

Для получения частичного балла при $2 \leqslant a_i \leqslant 4$ достаточно было вручную исследовать, какие ответы возникают на 7 возможных тестах этой группы, занести их в программу и выводить нужный ответ в зависимости от ввода.