Задача A. Геометрическая прогрессия

Прежде всего, необходимо отсортировать числа по возрастанию. Это возможно, поскольку даже если дана геометрическая прогрессия с $q < 1$, мы можем «перевернуть» последовательность, заменив знаменатель прогрессии на $\frac{1}{q} > 1$.

После сортировки будем считать, что $A \leqslant B \leqslant C$.

Существует несколько способов проверить, являются ли A, B, C последовательными членами геометрической прогрессии. Наиболее элегантный из них заключается в проверке равенства

$$B^2 = AC.$$

Задача B. Ключевые мутации

Любой из видов мутаций добавляет символы с одного из концов строки и, возможно, выполняет отрицание строки. Поэтому после каждой мутации строка содержит внутри себя версию до мутации или «отрицание» версии до мутации. Это наблюдение позволяет избежать создания новых строк. Обозначим через i и j позицию первого символа и позицию, следующую за последним символом в текущей версии, соответственно. Положим z равным 1 или 0, в зависимости от того, требуется ли или не требуется, соответственнно, для получения текущей версии ключа «отрицать» фрагмент с i-го до (исключая) j-го символа.

Тогда вначале $i = 0$, $j = N$, $z = 0$. Пусть введенная строка равна $a_0 a_1 \dots a_{N-1}$. Алгоритм, который, образно говоря, проигрывает мутации в обратную сторону, может выглядеть так:

```
пока i < j − 1:
    если (z = 0 и a_{j−3}a_{j−3}a_{j−1} = 011) или (z = 1 и a_{j−3}a_{j−3}a_{j−1} = 110), то:
        j := j − 1;
    иначе:
        если (z = 0 и a_i = 1 и a_{j−1} = 0) или (z = 1 и a_i = 0 и a_{j−1} = 1), то:
            i := i + 1;
            z := 1 − z;
        иначе:
            выйти из цикла
```

После выхода из цикла ответом будет являться фрагмент исходной строки с i-го до (исключая) j-го символа, если $z = 0$, или «отрицание» этого фрагмента, если $z = 1$.

Задача C. Треугольники

После ввода данных необходимо отсортировать массив длин отрезков a_i по возрастанию.

Затем нужно организовать тройной цикл для обхода всех вариантов длин сторон треугольников. Чтобы сократить перебор во вложенных циклах на втором и третьем уровнях нужно пропускать элементы, которые уже были обработаны на предыдущих уровнях. На Python это можно написать так:

```
for i, x in enumerate(lengths[:-2]):
    for j, y in enumerate(lengths[i + 1:-1], i + 1):
        for k, z in enumerate(lengths[j + 1:], j + 1):
            if x + y <= z:
                continue
```

Обозначим длины сторон в очередном треугольнике: x, y, z. Благодаря сортировке, условиям задачи и организации циклов можно быть уверенным, что $x < y < z$. Для каждого треугольника нужно проверить его существование.

Далее нужно составить два отношения между длинами сторон y/x и z/x, сохранить их в новом массиве и отсортировать его.

После этого перебираем пары соседних элементов в массиве отношений. Для подобных треугольников пары отношений будут совпадать. Если найдено M подобных треугольников, то число пар

треугольников, которые можно из них составить, равно $M(M-1)/2$. Сумма всех таких значений является ответом задачи.

В целом вычислительная сложность такого решения эквивалентна $O(N^3)$, что вполне укладывается в ограничения по времени при N, не превышающем 200.

Задача D. Командная олимпиада

Для первой подзадачи ответом является минимум из k и $b_1 - a_1$, поскольку мы максимально можем сократить неравномерность на k, но верхним пределом является $b_1 - a_1$.

Для второй подзадачи достаточно просто вычислить искомое значение. С помощью цикла проходимся по командам и прибавляем к ответу $\max(a_i, b_i, c_i)$ - $\min(a_i, b_i, c_i)$.

Для третьей подзадачи можно k раз пройтись по всем командам, выбрать такую команду, у которой $b_i - a_i$ максимально, тренировать a_i и в дальнейшем не рассматривать эту команду.

Для четвертой подзадачи из-за ограничений нам не важен порядок тренировки самых слабых членов команд. В результате оказывается, что к значению из первой подзадачи надо просто прибавить сумму $k + (k-1) + (k-2) + ... + (k - min(k, n-1))$. Данная сумма находится либо формулой арифметической прогрессии, либо циклом от k до $k - min(k, n-1)$.

Для пятой подзадачи для каждой команды надо вычислить $b_i - a_i$, который мы определим как отставание a_i. Отсортируем полученные значения по убыванию. Далее, тренируем очередного самого слабого члена команды в размере его отставания a_i, либо, если наш k упал ниже отставания a_i, в размере текущего k. Порядок убывания отставаний a_i правильный. Доказательство следующее: если мы сначала тренируем ученика, у которого отставание меньше, чем у ученика, которого мы тренируем позже, то несложно заметить, что если мы поменяем их порядки, то не может быть такого, что их общее отставание увеличилось, но может быть такое, что их общее отставание сократилось. В конце получаем порядок тренировки учеников с уровнем знаний a_i, отставания которых убывают.

Задача E. Баланс работа-жизнь

В первой подзадаче достаточно просто вывести 2, поскольку дополнительное ограничение гарантирует нам то, что Коля всё успеет.

Во второй подзадаче надо проверить входные данные на соответствие одному из 3 описанных случаев. Иными словами:

- выводим 0, если $a > n$;
- выводим 1, если $a \leqslant n$, но $a + b > n$;
- выводим 2, если $a + b \leqslant n$.

Задача F. У кого больше

Рассмотрим алгоритм определения максимального количества цифр в разложении числа n в виде суммы последовательных натуральных чисел. Сумма последовательных k натуральных чисел начинающихся числом a равна

$$a + (a+1) + \cdots + (a+k-1) = ak + k(k-1)/2.$$

Приравняем полученную сумму n, имеем $ak + k(k-1)/2 = n$. Тогда

$$a = (n - k(k-1)/2)/k.$$

Так как a должно быть целым, то вначале найдем максимальное k, при котором a положительно:
C++:

```
k = 1;
while (n-k*(k-1) / 2 > 0) k++;
if (n-k*(k-1) / 2 != 0) k--;
```

Python:

```
k = 1
while n-k*(k-1) // 2 > 0: k = k+1
if n-k*(k-1) / 2 != 0: k = k-1
```

Далее, уменьшая k, найдем, при каком k число a будет целым:
C++:

```
while ((n-k*(k-1) / 2) % k > 0) k--;
```

Python:

```
while ((n-k*(k-1) / 2) % k > 0): k = k-1
```

Используя данный алгоритм для чисел x и y, найдём ответ на задачу.

Задача G. Сила Архимеда

Перенесем все члены неравенства в левую сторону:

$$(F_{i_1} - m_{i_1}g) + (F_{i_2} - m_{i_2}g) + \ldots + (F_{i_k} - m_{i_k}g) > 0.$$

Тогда заметим, что можно просто жадно брать шары по убыванию значений $F_i - m_i g$.

Задача H. Игровые площадки на поле

После чтения данных программа ищет в прямоугольном поле $N \times M$ все 2×2 блоки, состоящие только из нулей, которые могут быть преобразованы в игровые зоны. Она отмечает эти области в дополнительной матрице b.

Наконец, программа использует Depth-First Search (DFS) для поиска и подсчета связных областей в матрице b. Каждая связная область представляет собой одну большую игровую площадку. Результатом является количество таких площадок, которое выводится на экран.