Задача A. Чтение в дороге

Опишем решение задачи на полный балл.

Чтобы определить максимальное время прочтения всех статей, нужно предположить, что Кеша тратил на каждую следующую статью на единицу больше времени. Это следует из утверждения, что для каждой пары статей, прочитанных одна за другой, Кеша либо потратил на чтение обеих статей одинаковое время, либо потратил на вторую статью пары время, на единицу большее, чем потратил на первую статью пары.

Тогда можем считать, что на первую прочитанную статью он потратил единицу времени, на вторую прочитанную статью — две единицы и т.д. На последнюю прочитанную статью, соответственно, он потратит n единиц времени. Номера статей в этом случае не имеют значения, затраченное на их чтение время будет равно сумме чисел от 1 до n. Эта сумма может быть вычислена либо соответствующим циклом, либо по формуле Гаусса $1 + 2 + \ldots + n = (n + 1) \cdot n/2$.

Для вычисления минимального времени нужно предположить, что Кеша тратил на следующую статью на единицу больше времени только в том случае, если не мог потратить столько же.

Нам известно следующее. Во-первых, Кеша всегда выбирал для прочтения самую короткую статью. Это означает, что для любой пары статей, прочитанных одна за другой, длительность чтения первой статьи пары не превосходит длительности второй статьи пары.

Во-вторых, если статьи требуют одинакового времени на чтение, то Кеша сначала читает статью с меньшим номером. Это означает, что для любой пары статей, прочитанных одна за другой, длительность чтения первой статьи пары будет совпадать с длительностью чтения второй статьи пары только в том случае, если номер первой статьи пары меньше, чем номер второй статьи пары. Если же номер первой статьи пары больше номера второй статьи пары, то длительность чтения второй статьи пары на единицу больше, чем длительность чтения первой статьи пары.

Таким образом, мы можем анализировать два идущих подряд номера статей и увеличивать время, затраченное на прочтение очередной статьи, если ее номер меньше номера предшествующей ей статьи.

Фрагмент кода на Python:

```
minimumTime = 0
previousArticleTime = 1
previousArticleNumber = 0
for i in range(n):
    a = int(input())
    if a < previousArticleNumber:
        previousArticleTime = previousArticleTime + 1
    minimumTime = minimumTime + previousArticleTime
    previousArticleNumber = a
```

При использовании $C++$, $Java$, $C\#$ и других строго типизированных языков следует учитывать, что, по крайней мере, тип переменных $maximumTime$ и $minimumTime$ должен быть 64-битным целочисленным: в противном случае решение не пройдёт последнюю группу тестов.

Задача B. Штрафные очки

Опишем решение задачи на полный балл.

Сначала определим, какие задачи решил Кеша. Для этого отсортируем последовательность попыток по неубыванию номеров задач. Кеша решил первые k задач из этой отсортированной последовательности, поскольку известно, что если он решил задачу с номером X, то он решил и все задачи с меньшими номерами (если он делал по ним хотя бы одну попытку).

Теперь нужно определить, сколько попыток всего было сделано по этим первым k задачам, и вычесть из полученного результата число k: очевидно, что Кеша совершил k успешных попыток, сдав k задач. Также можно просто не добавлять к ответу одну попытку по каждой учитываемой задаче. Именно такой подход реализован в приведённом фрагменте кода: первая попытка по каждой учитываемой задаче не добавляется к ответу ($answer$).

Фрагмент кода на Python (в списке $attempts$ хранятся номера задач в той последовательности, в которой Кеша делал по ним попытки):

```
answer = 0
previous = 0
count = 0
attempts.sort()
for p in attempts:
    if p == previous:
        answer = answer + 1
    else:
        previous = p
        count = count + 1
        if count == k+1:
            break
print(answer)
```

Подзадачи 1 и 2 допускали квадратичные алгоритмы решения (использование далеких от оптимальности методов сортировки), в подзадаче 1 можно было даже перебрать различные варианты наборов задач, которые мог решить Кеша. Позадача 3 сводилась к отысканию минимального номера задачи в заданной последовательности.

Для решения задачи на полный балл также можно было воспользоваться структурой данных, поддерживающей сортировку (например, $TreeMap$ или $TreeSet$ в Java и их аналоги в других языках).

Задача C. Музыкальный выбор

При решении этой задачи можно было действовать разными способами, так или иначе перебирая варианты и, возможно, отбрасывая неподходящие, не рассматривая их до конца. Ограничения задачи вполне позволяли полностью перебрать все возможные варианты для каждого тестового случая.

Один из способов перебрать все возможные варианты состоял в генерации всех перестановок из имен участников. Далее для каждой перестановки выполнялась проверка, позволяет ли данная перестановка каждому участнику получить один из желаемых призов.

Алгоритмы генерации перестановок описаны в различных источниках; ряд языков предоставляет их как стандартный алгоритм.

Задача D. Диапазоны

Опишем решение задачи на полный балл.

Известно, что номера участников не превосходят 10^6. Опишем список (массив) $participant$ соответствующего размера, элемент списка $participant[i]$ будет содержать количество задач, решенных участником с номером i. Обновлять этот список просто: когда очередной участник сдаёт задачу (становится известен его номер), соответствующий элемент массива увеличивается на 1.

Также опишем ещё один список $sumProblems$, элемент списка $sumProblems[j]$ содержит количество участников, которые решили j и более задач. В качестве размера этого списка можно использовать максимально возможное количество запросов (оно не превосходит $3 \cdot 10^5$; очевидно, что количество задач, которое мог решить один участник, не может быть больше).

Зная элементы этого списка, мы всегда сможем вычислить, какие места занимают участники, сдавшие некоторое количество задач x. Как можно понять, $sumProblems[x]$ даёт последнее место, которое занимает участник с таким количеством задач. А первое место, которое занимает такой участник, будет определяться как $sumProblem[x + 1] + 1$.

Когда очередной участник с номером i сдает задачу, нужно узнать, какая это по счёту задача у участника — а это определяется элементом $participant[i]$, после чего увеличить соответствующий элемент $sumProblems[participant[i]]$ на единицу. Никакие другие элементы изменениям не подвергнутся.

Когда же приходит запрос от Прохора, нужно выяснить, сколько задач сдал участник с номером 1 (Кеша) (пусть y задач) и сообщить искомый диапазон $sumProblems[y + 1] + 1$ $sumProblems[y]$.

Неполный балл мог быть набран различными симуляциями процесса.

Задача E. B + C

Для длины чисел $\leqslant 3$ достаточно было перебрать два числа (например B и C) в промежутке $1 \dots 10^3$ и проверить каждую пару на соответствие условию.

В общем случае для решения предлагается использовать принцип динамического программирования.

Первым делом, если $|B| > |A|$ или $|C| > |A|$, то ответ, очевидно, 0.

Будем нумеровать позиции в строке справа-налево (в порядке возрастания номеров разрядов), а не слева-направо.

Рассмотрим функцию $dp(pos, carry)$ — количество способов восстановить разряды в полуинтервале $[0; pos)$ таким образом, что соблюдается равенство при сложении и перенос в следующий разряд равен $carry$ (он может быть равен либо 0, либо 1).

Обозначим длину числа A через N. Ответ в таком случае будет равен $dp(N, 0)$ — востановили все N разрядов и нет переноса «в пустоту».

Для вычисления $dp(pos, carry)$ переберём все возможные варианты цифр в разряде pos для чисел B и C (в худшем случае их будет $10 \cdot 10$ на разряд), а также «старый» перенос $carry_{old}$.

Для каждой пары цифр bd и cd вычислим сумму $as = (bd + cd + carry_{old})$. В таком случае ожидаемая цифра в числе A будет равна $ad = as \mod 10$, а перенос в новый разряд $carry$ будет равен целочисленному частному от деления as на 10.

Соответственно, если ad соответствует входной строке A (либо та же цифра, либо #), то к значению $dp(pos, carry)$ добавляется значение $dp(pos - 1, carry_{old})$.

Надо было не забыть, что в числах запрещены ведущие нули, поэтому на самых старших разрядах чисел B и C перебор следовало начинать с 1.

Так как в общем случае ответ может быть очень большим (порядка 10^{200}), то необходимо было использовать арифметику остатков и после каждого (желательно) умножения брать остаток от деления на $10^9 + 7$.

Приведем решение на Python.

```
def solve():
    nums = list(input() for _ in range(3))
    a, b, c = (num[::-1] for num in nums)

    unknown = '#'

    def get_ans():
        base = 10
        mod = 10**9 + 7

        n = len(a)

        if n != max(map(len, nums)):
            return 0

        dp = [[0] * (n + 1) for _ in range(2)]

        dp[0][0] = 1

        def lims(num, ind):
            if len(num) <= ind:
                return 0, 1

            letter = num[ind]

            if unknown == letter:
```

```
                start = int(ind + 1 == len(num))
                return (start, base)

            digit = int(letter)
            return digit, digit + 1

        for i in range(n):

            aStart, aEnd = lims(a, i)
            bStart, bEnd = lims(b, i)
            cStart, cEnd = lims(c, i)

            for bd in range(bStart, bEnd):
                for cd in range(cStart, cEnd):
                    for carry in range(2):
                        a_sum = bd + cd + carry

                        ad = a_sum % base
                        next_carry = a_sum // base

                        if aStart <= ad < aEnd:
                            dp[next_carry][i + 1] += dp[carry][i]
                            dp[next_carry][i + 1] %= mod

        return dp[0][-1]

    print(get_ans())

if __name__ == "__main__":
    solve()
```