Задача A. Крестики-нолики

Для поля 3x3 ответ - да (ничья), в осталных случаях - нет. Можно рассмотреть все случаи и убедиться в этом.

Задача B. Олимпиадная задача

Если $k = 7$, ответ равен 7. При $k \neq 7$ можно заметить, что число, составленное из n цифр k, делится на 7 тогда и только тогда, когда на 7 делится число, составленное из n единиц. Поэтому при $k \neq 7$ ответ равен $k \cdot 111\,111$. Если вы не догадались до этого факта, можно было написать цикл, добавляющий по одной цифре в конец числа и проверяющий, делится ли результат на 7.

Задача C. Квадратное уравнение

Для решения этой задачи нужно было вспомнить теорему Виета, по которой числа m/n и p/q являются корнями квадратного уравнения $x^2 - (m/n + p/q) \cdot x + (m/n \cdot p/q) = 0$. Помножив коэффициенты этого уравнения на $n \cdot q$, получим уравнение с целыми коэффициентами $(n \cdot q) \cdot x^2 - (m \cdot q + p \cdot n) \cdot x + (m \cdot p) = 0$. Ответ задачи (один из возможных): $a = n \cdot q$; $b = -m \cdot q - p \cdot n$; $c = m \cdot p$.

Важно, что при вычислениях нельзя использовать 32-битный целый тип — это приведёт к Wrong Answer на 10-м тесте из-за переполнения.

Задача D. Аккорды

Решаем итеративно. Для нечётных строк делаем сплит по пробелам и получаем аккорды. Чётные просто читаем и выводим без изменений.

Чтобы преобразовывать аккорды, можно завести массив со всеми аккордами и словарь, в котором ключом будет аккорд, а значением — его индекс в массиве. Для каждого аккорда получаем его индекс, сдвигаем его на X по модулю 12 (это число всех аккордов) и с помощью массива получаем новый аккорд.

Задача E. Суффиксный массив

Если два подряд идущих символа различаются, то наибольший общий префикс начинающихся с них суффиксов имеет длину 0. Таким образом, ответ не равен нулю только внутри блока из одинаковых букв.

Нетрудно видеть, что внутри блока из k букв ответ будет состоять из чисел $k-1$, $k-2$, …, 2, 1. Таким образом, чтобы решить задачу, нужно выделить все блоки подряд идущих чисел, и вывести для каждого убывающую последовательность натуральных чисел до 1, разделив их нулями.

Альтернативное решение — идти по строке с конца, поддерживая счётчик количества одинаковых букв подряд. Для каждой буквы, если она отличается от предыдущей, счётчик обнуляется; иначе, счётчик увеличивается на единицу. Нужно вывести все значения счётчика в обратном порядке.

Задача F. Серия пенальти

Сразу разобьём задачу на два случая: $r \geqslant 6$ и $r \leqslant 5$, которые будем решать по-разному.

В первом случае серия продлилась дольше пяти пар ударов. Это означает, что после пяти пар ударов был равный счёт, а в дальнейшем количество забитых командами голов может различаться только на один. Значит, если $r \geqslant 6$, то решение существует только при $f = r - 1$ (пример серии — обе команды забили по $r - 1$ разу, далее Россия забила, а Франция не забила, при этом неважно, кто бил первым).

При $r \leqslant 5$ остаётся только 15 допустимых комбинаций $r : f$. Можно перебрать их все на бумажке, нарисовав для каждой возможную серию пенальти (и потом вбить эту серию в свою программу) или поняв, что её не существует. Не существует серий пенальти для счёта 5:0, 5:1, 5:2 и 4:0, для остальных 11 комбинаций ответ есть (и часто не единственный).

Задача G. Манхэттенские полицейские

Достижимые клетки образуют на сетке «ромб», верхней клеткой которого является клетка $(x - k; y)$, нижней — $(x + k; y)$. Давайте пройдёмся циклом по строкам ромба от $max(0, x - k)$ до $min(x + k, m)$ (чтобы не выйти за границы).

Известно, что если мы находимся в строке i, то мы уже накопили расстояние $dx = |x - i|$ от стартовой позиции. Значит, можем влево или вправо от y отойти на расстояние не большее $k - dx$. Следовательно, в строке i нас устраивают все клетки от $y - (k - dx)$ до $y + (k - dx)$.

Чтобы не выходить за границы, определим диапазон достижимых клеток в строке i: $left = max(0, y - (k - dx))$, $right = min(n - 1, y + (k - dx))$. Тогда к ответу можно прибавить количество чисел на отрезке от $left$ до $right$, то есть $right - left + 1$.

Задача H. Эхолокация

Обратим внимание, что координаты вершин от 0 до 1000. Таким образом, мы имеем дело с квадратным полем 1000×1000, каждая клетка 1×1 которого либо целиком внутри строения, либо целиком снаружи. При этом для каждой вертикальной стены либо все клетки смежные ей слева находятся внутри, а все смежные ей справа — снаружи, либо наоборот.

Посортируем стены по координате x и воспользуемся алгоритмом заметающей прямой, чтобы узнать для каждой клетки, внутри она или снаружи. Изначально вертикальный столбец из клеток находится целиком снаружи. Каждый следующий столбец совпадает с предыдущим, но вертикальные стены инвертируют состояния клеток: если клетка в некотором ряду была снаружи, то после вертикальной стены она окажется внутри, и наоборот. Таким образом мы может полностью восстановить все клетки внутри многоугольника.

Альтернативное решение заключается в восстановлении границы многоугольника по вертикальным отрезкам. Для этого надо заметить, что вершин с некоторой фиксированной координатой y чётное количество, и они соединены горизонтальными сторонами. Нужно сгруппировать вершины по координате y, посортировать по координате x и соединить стороной 1-ю и 2-ю, 3-ю и 4-ю, ..., $(2n-1)$-ю и $2n$-ю вершины. После чего можно явно обойти многоугольник и посчитать его площадь.

Задача I. Подарок для Вовы

Рассмотрим мультиграф $G(s)$, построенный на буквах исходной строки s, ребрами которого являются элементы множества $M(s)$. Заметим, что так как $|s| > 1$, то для любой строки t у которой $M(t) = M(s)$ верно, что $G(t) = G(s)$.

Так как строка s задаёт в графе $G(s)$ некоторый эйлеров путь, то задача состоит в том, чтобы найти лексикографически минимальный эйлеров путь в графе $G(s)$. Для решения этой задачи достаточно внести небольшие модификации в достаточно эффективный алгоритм поиска эйлерова пути в ориентированном графе.

Рассмотрим алгоритм Флёри, который последовательно строит эйлеров путь в графе, каждый раз вычеркивая произвольное ребро исходящее из текущей вершины, удаление которого не нарушает связности текущего графа (не считая изолированных вершин). Чтобы данный алгоритм построил лексикографически минимальный эйлеров путь достаточно перебирать ребра из текущей вершины в алфавитном порядке увеличения значения букв концов ребёр. Данный алгоритм работает за $O(nA^2)$ и его наивная реализация на `C++` укладывается в `TimeLimit`.

Также можно модифицировать стандартный алгоритм поиска эйлерова путь на основе циклов за $O(n)$. Реализация данного алгоритма спокойно укладывается в `TimeLimit` на языке `Python`.

Особое внимание нужно уделить случаю, когда первая и последняя буквы строки s совпадают. В этом случае в $G(s)$ есть эйлеров цикл и перед запуском алгоритма поиска цикла нужно выбрать в качестве стартовой вершины минимальную букву, встречающуюся в s.

Задача J. Приличные слова

Заметим, что нам не важен порядок букв в слове, а важно только наличие или отсутствие какой-то буквы. Тогда каждое слово можно представить в виде 26-битного числа, где i-й бит равен 1, если буква i есть в слове и 0 иначе. Назовем получившееся число битовой маской слова. Теперь, чтобы проверить, есть ли нужная строка в наборе слов, достаточно проверять, есть ли в наборе слов битовая маска такой строки. В наборе слов некоторые слова могут иметь одинаковую маску, тогда всегда выгодно оставлять только то слово, стоимость которого меньше.

Теперь, чтобы разбить строку на подстроки, заведем динамику. dp_i - минимальная стоимость разбиения первых i символов строки (∞, если разбиение невозможно). Изначально $dp_0 = 0$, остальные $dp = \infty$. Теперь для каждой позиции i будем пытаться приписать в конец строку длины j и

улучшить тем самым ответ для dp_{i+j}. Нет смысла пытаться приписывать строку длиннее 26 символов, потому что в ней заведомо какая-то буква встретится хотя бы два раза и в наборе слов не найдется подходящего слова. Последовательно будем увеличивать j и поддерживать битовую маску приписываемой строки. Тогда можно легко проверять, что в текущей строке никакая буква не встречается больше одного раза и заодно с помощью структуры данных типа **map** или **dict** искать в наборе слов минимальную стоимость слова с такой маской. В конце ответ будет содержаться в $dp_{|s|}$

Задача K. Тараканы

Для каждой точки посчитаем, сколько раз она встретилась во входных данных. Это можно] сделать с помощью структуры данных типа **map** или **dict**. После чего для каждой точки посчитаем, во сколько прямоугольников она входит. Если эти два числа для некоторой точки не совпадают, то точка является тараканом, и нужно прибавить количество её вхождений в ответ.

Считав все данные, предварительно сделаем сжатие по координатам (сжатия по координате y достаточно), таким образом координаты y будут от 0 до 89999. После чего воспользуемся алгоритмом заметающей прямой: для каждой координаты y будем поддерживать количество прямоугольников, покрывающей её. Левая граница прямоугольника будет увеличивать это значение на 1 на отрезке, а правая граница будет уменьшать на 1. Встретив точку в порядке обхода заметающей прямой, сделаем запрос поддерживаемого значения в одной координате y. Выполнять эти операции можно, например, **деревом Фенвика**, в котором хранятся префиксные суммы.