Задача A. За гробоцветами

Как известно, две различные прямые не имеют двух общих точек. Рассмотрим первых трех охотников. Если они лежат не на одной прямой, то ответ найден, иначе рассмотрим 2-го, 3-го и 4-го охотника. Если эти 3 человека лежат на одной прямой, то все четыре охотника находятся на одной прямой (у них две общие точки 2 и 3 охотник). Значит можно просто перебрать трех подряд идущих охотников. Если какие-то три не лежат на одной прямой, то ответ найден, иначе ответа не существует (все лежат на одной прямой).

Задача B. Сильная группа

Посмотрим на множество, которое является оптимальным ответом. Если его диаметр (максимальный по длине путь) имеет хотя бы 3 ребра, то удалим среднее в этом диаметре. Множество распалось на два связных, в каждом из которых находится хотя бы две вершины. Тогда хотя бы в одном из них среднее значение будет не меньше. Значит в одном из оптимальных ответов длина диаметра не более двух.

Если длина диаметра 1, то это просто две соседних вершины. Если же длина диаметра равняется двум, то такой граф выглядит как несколько вершин, соединённые с одной центральной. Для каждой центральной переберём количество соседей, которые мы возьмём. Ясно, что выгодно брать максимальные по полезности вершины. Тогда для каждой вершины отсортируем соседей по убыванию полезностей, и переберём, какой префикс получившегося массива мы возьмём. Среди всех найденных средних значений найдём максимальное. Это и будет ответом.

Задача C. Ловушка со свечками

Если начальная или конечная строка состоит из одинаковых символов, или же она имеет чётную длину, и в ней чередуются два символа, то её невозможно получить ни из какой другой и из неё невозможно получить никакую другую. Поэтому в этом случае преобразование возможно, лишь если начальная и конечная строки совпадают: тогда можно просто ничего не делать.

В остальных случаях преобразование всегда возможно. Найдём в первой строке строке символ s_i, отличный от двух его соседей (если такого нет, то это случай из предыдущего абзаца). Начиная с него и идя по часовой стрелке, будем каждый символ s_j заменять на любой символ, отличный от s_{j-2} и s_{j+2} (индексы рассматриваются по модулю n). При $j = i$ мы знаем, что вокруг s_j различные символы, а при остальных j, делая непосредственно перед этим замену символа s_{j-1}, мы сделали так, чтобы он отличался от s_{j+1}. После этих n операций мы добились того, что каждая пара символов, стоящих через один, различна.

Теперь найдём во второй строке символ t_k, у которого различны соседи (опять же, такой есть). Начиная с $j = k+1$, мы будем заменять каждый символ s_j на любой символ, отличный от t_{j-2} и s_{j+2}. Снова поймём, что так можно делать: во время первой операции все пары символов, стоящих через один, различны, поэтому можно изменить любой символ, а во время каждой следующей операции s_j можно поменять, так как прямо перед этим s_{j-1} был заменён на какой-то символ, не равный s_{j+1}. Вторым проходом мы добились, что каждый символ s_j не равен t_{j-2}. Кроме того, последний символ, который мы изменяли, то есть s_k, не равен s_{k+2}.

Последним, третьим проходом, мы будем, начиная с символа s_{k+1}, заменять все символы s_j на t_j. Первую такую операцию над символом s_{k+1} можно сделать, так как s_k и s_{k+2} различны. Каждая следующая операция над очередным символом s_j возможна, так как по построению из предыдущего абзаца символ s_{j+1} отличается от $t_{j-1} = s_{j-1}$ (ведь в течение третьего прохода символ s_{j+1} мы ещё поменять не успели). Исключением является лишь символ s_k, ведь, дойдя до него, мы уже успели поменять символ s_{k+1} на t_{k+1}. Тем не менее, мы сможем поменять s_k на t_k, поскольку $s_{k-1} = t_{k-1} \neq t_{k+1} = s_{k+1}$.

Таким образом, за $3n$ операций мы точно сможем из одной строки получить другую.

Задача D. Чары

Заметим, что так как произведение чисел от a до b положительное, ответом могут быть только числа от 1 до 9. Так как все эти числа дают разный остаток по модулю 9, а также остаток числа по модулю 9 равен остатку суммы цифр числа по модулю 9, можно найти произведение чисел от a до b по модулю 9, и вместо 0 вывести 9.

Однако, мы всё ещё не можем перебрать все числа от a до b. Поэтому заметим, что если $a+9 \leqslant b$, то среди чисел от a до b точно есть число делящееся на 9, значит и произведение будет делиться на 9, значит ответом будет 9.

Осталось научиться проверять, правда ли, что $a + 9 \leqslant b$, и если нет, то находить остаток от деления длинного числа на 9 и прибавлять к длинному числу 1. Для этого можно воспользоваться языком программирования, в котором есть длинная арифметика, либо реализовать нужные функции самостоятельно.

Задача E. Ленивые лесорубы

Заметим, что набор отрезков подходит, то есть уменьшает высоту на каждом единичном отрезке на целое число метров, если каждый единичный отрезок покрывается четным числом отрезков. Для этого, необходимо и достаточно, чтобы каждая точка встречалась как конец отрезка чётное число раз. Докажем это:

- Пусть есть набор отрезков, при котором каждая точка встречается как конец отрезка чётное число раз. Пусть есть единичные отрезки, которые покрыты нечётным числом отрезков. Рассмотрим самый левый из них, пусть это отрезок $[x, x + 1]$. Тогда соседний слева отрезок, $[x - 1, x]$, покрыт чётным числом отрезков. Множество отрезков, покрывающих $[x, x + 1]$, получается из множества отрезков, покрывающих $[x - 1, x]$, удалением отрезков, закончившихся в x, и добавлением отрезков, начавшихся в x. Иными словами, $n_{[x,x+1]} = n_{[x-1,x]} - e_x + b_x$. Мы знаем, что $n_{[x-1,x]}$ и $(e_x + b_x)$ — чётны. По модулю 2, сложение эквивалентно вычитанию. Значит, $(e_x - b_x)$ — тоже чётно. Поэтому, $n_{[x,x+1]}$ — чётно. Противоречние.
- Необходимость доказывается аналогично. Нужно рассмотреть минимальное x, встречающееся как конец отрезка нечётное число раз.

Теперь мы свели задачу к такой: дан массив пар чисел, посчитать количество отрезков массива, на которых каждое число встречается четное число раз. Заменим все числа таким образом, чтобы их значения стали целыми числами от 0 до $k-1$, где k — количество различных чисел. Будем идти слева направо и для текущего префикса поддерживать двоичный вектор $pref$ длины k, $pref_i$ равно количеству раз, которое встречается число i на текущем префиксе, по модулю 2. Все значения анологичного вектора, построенного для отрезка, являющегося ответом, равны 0. Вектор для отрезка $[l, r]$ равен поэлементному xor-у векторов для префикса r и префикса $l - 1$. Поэтому, для каждого префикса нужно прибавить к ответу количество предыдущих префиксов, в которых вектор совпадал с текущим. Для этого можно вместе с вектором поддерживать хеш вектора. При добавлении очередной пары, хеш пересчитывается за $O(1)$. Количество префиксов с каждым значением хеша можно хранить в хеш таблице.

Задача F. Арифметика и кубики

Посчитаем для каждой цифры число кубиков, на которых она встречается хотя бы один раз.

Забавно, но этой информации достаточно чтобы почти точно получить ответ.

Наблюдение 1. Если каждая цифра от 1 до 9 встречается хотя бы n раз, то можно выложить любое число длины n и меньше, не содержащее нулей в записи.

Доказательство. Выложим первую цифру числа любым подходящим кубиком. Не сложно видеть, что теперь каждая цифра встречается хотя бы $n - 1$ раз. Повторим данное рассуждение ещё $n - 1$ раз.

Наблюдение 1^+. Если каждая цифра от 1 до 9 встречается хотя бы n раз, а 0 встречается хотя бы $n - 1$ раз, то можно выложить любое число длины n или меньше.

Доказательство$^+$. Дословно применить предыдущее доказательство не получится: может оказаться, что в момент когда мы уже выложили $n - 1$ цифру из n, то все нули закончились, а нужно выложить ноль. Давайте сначала выложим все нулевые цифры, а потом остальные. Тогда такой проблемы не возникнет.

Оказывается, что если n — минимальное число, не подходящее под условие леммы, то ответ является числом длины n, более того, он выглядит как одно из:

- $d000\ldots00$,
- $dddd\ldots dd$, где d какая-то цифра, а длина всего числа n.

Поймём почему. Начнём с того, что поймём, чего же не хватило, чтобы увеличить n. Пусть нулей. Тогда $1000\ldots000$ является ответом.

Пусть нулей достаточно, а наименьшая цифры, которой не хватило — это d. Кстати несложно видеть, что любое число длины n только из цифр $0, 1, \ldots, d-1$ всё ещё можно представить. А вот число $dddddd\ldots dddd$ уже точно не представить.

Число $d000\ldots00$ иногда тоже может являться ответом. Хотя и нулей и цифры d достаточно много (обоих хотя бы по $n-1$), но если их обоих ровно $n-1$ и они находятся на одинаковых кубиках, то собрать $d000\ldots00$ не выйдет.

Может быть неочевидно, почему ответом не может быть что-то промежуточное: сколько-то d и сколько-то 0. Но всё достаточно просто, покажем, что если мы умеем выражать $d000\ldots000$, то можно и любое другое число из нулей и d, кроме всех d. Возьмём ответ для $d0000\ldots0000$. У нас есть достаточно много кубиков содержащих d и не использованных сейчас для отображения d. Если этот кубик показывает сейчас 0, то просто перевернём его. Если этот кубик не используется, просто заменим один из кубиков с 0 на него.

Задача G. Счёт в теннисе

Будем решать задачу методом динамического программирование. Причем, будем искать ответ с конца. То есть, вычислять путь минимальной длины от (a, b) до $(0, 0)$, а не наоборот. Для пересчёта можно использовать формулу $dp_{a,b} = \min(dp_{a-1,b}, dp_{a,b-1}) + \gcd(a, b)$. Однако, это решение пока работает за $O(a \cdot b)$. Попробуем его ускорить. Не умаляя общности, $a \leqslant b$. Пусть $a < b$ и b — простое. Тогда вес минимального пути от $(0, 0)$ до (a, b) равен $a + b$. Понятно, что меньше он быть не может, потому что после каждого хода вес увеличивается хотя бы на 1. А для того, чтобы его достичь, можно действовать по следующему алгоритму:

- $(0, 0)$
- $(1, 0)$
- $(1, x)$, где x увеличивается от 1 до b. При этом, после каждого изменения НОД равен 1, так как одно из чисел равно 1
- (y, b), где y увеличивается от 2 до a. При этом, после каждого изменения НОД равен 1, так как $y < b$ и b — простое

Также, если $a = 0$, от $(0, 0)$ до $(0, b)$ существует единственный путь, и его вес равен $\sum\limits_{x=1}^{b} x = \frac{(b+1)\cdot b}{2}$.

Будем вместо стандартного динамического программирования реализовывать перебор с запоминанием, который ещё называют ленивым динамическим программированием. Если при таких a и b сделать отсечение и сразу вернуть ответ за $O(1)$, решение будет работать примерно за $O(a \cdot \log(b))$. Это следует из того, что простые числа в среднем встречаются раз в логарифм чисел. И на практике нет большого разрыва между соседними простыми числами до 10^9.

Осталось научиться отсекать перебор по a. По аналогии, хотелось бы сделать это, когда a стало простым числом. Однако, возможна ситуация, при которой a — простое, $a < b$, но вес минимального пути от $(0, 0)$ до (a, b) не равен $a + b$. Поэтому, нужно добавить дополнительное условие. Пусть p — максимальное простое число, что $p \leqslant b$. Тогда, если дополнительно $a < p$ и на отрезке $[p, b]$ нет чисел делящихся на a, вес минимального пути от $(0, 0)$ до (a, b) точно равен $a + b$. Для его достижения можно действовать по следующему алгоритму:

- $(0, 0)$
- $(1, 0)$
- $(1, x)$, где $x \in [1, p]$

- (y, p), где $y \in [2, a]$
- (a, z), где $z \in [p+1, b]$. При этом, после каждого изменения НОД равен 1, так как a — простое, $a < z$ и z не делится на a

Таким образом, мы не всегда отсекаемся, когда a первый раз стало простым, но на одном из ближайших простых мы точно отсечёмся. На практике, такой перебор с отсечениями посетит максимум $\sim 10^5$ различных состояний, при ограничениях до 10^9.

Задача H. Свадьба

Для решения задачи рассмотрим независимо каждый двоичный бит.

По мере поступления запросов, будем поддерживать масссив $c_0, c_1, \ldots, c_{29}$, где c_i — количество фей на свадьбе, в двоичной записи значения общительности которых присутствует i-й бит (биты мы пронумеруем с 0 до 29 в порядке от младщих битов к старшим). Заметим, что если после каждого запроса мы будем поддерживать актуальные значения этого массива, по ним легко можно восстановить суммарную общительность всех фей: $\sum_{i=0}^{29} c_i \cdot 2^i$.

При появлении новой феи достаточно перевести значение ее общительности в двоичную систему и увеличить на единицу соответствующие значения c_i.

Когда к общительности каждой феи применяется операция побитового исключающего ИЛИ с числом v, необходимо заменить значения c_i, которые соответствуют единичным битам в двоичной записи числа v, так как во всех числах значения в этих битах изменится на противоположное. Если на текущий момент на свадьбе присутствует M фей, то необходимо заменить значения c_i на $M - c_i$.

При уходе одной из фей, нам нужно перевести ее текущее значение общительности в двоичную систему и уменьшить на единицу соответствующие значения c_i. Для того, чтобы определить значение общительности этой феи, необходимо взять исходное значение ее общительности и применить к нему все операции третьего типа, которые произошли после появления этой феи на свадьбе. Чтобы быстро применить все эти операции, будем поддерживать в каждый момент времени число X — значение побитового исключающего ИЛИ всех значений v из операций третьего типа. Для каждой феи запомним значение X на момент ее появления. Теперь, при удалении феи, достаточно взять текущее значение X и применить к нему побитовое исключающее ИЛИ со значением X в момент появления этой феи. Таким образом, мы получим исключающее ИЛИ значений v из всех запросов третьего типа за время, которая фея была на свадьбе.

Задача I. Защитный барьер

Будем решать задачу с помощью метода динамического программирования.

$dp_{l,r}$ будет означать множество множеств возможных коэффициентов, которые стоят при числах. (Коэффициент означет, сколько раз мы прибавили это число к ответу).

Переберем позицию максимума, тогда мы точно знаем какой при этом числе стоит коэффициент (количество отрезков, которые содержат этот элемент), а также знаем, какие отрезки пойдут влево и вправо, и можем пересчитать динамику перебрав коэфиценты из $dp_{l,m-1}$ и $dp_{m+1,r}$.

Чтобы найти ответ, не нужно знать порядок коэффициентов в массиве, поэтому количество различных состояний динамики ограничено количеством разбиений m на слагаемые (а более точно, количеством разбиений количества отрезков, целиком лежащих внутри состояния).

Таким образом время работы составит не более $O(n^3 \cdot P(m))$, но на самом деле гораздо меньше.

Задача J. Малефисумма

Давайте рассмотрим каждую тройку индексов (i, j, k) в сумме и зафиксируем j. Для фиксированного j в сумму входят произведения по всем $i < j$ и всем $k > j$.

Иными словами, перепишем нашу сумму как:

$$\sum_{j=1}^{n} \left(a_j \cdot \sum_{i<j,k>j} a_i \cdot a_k \right)$$

Поскольку выбор i и k не зависят друг от друга, можно переписать это как

$$\sum_{j=1}^{n} \Big(a_j \cdot (\sum_{i=1}^{j-1} a_i) \cdot (\sum_{k=j+1}^{n} a_k) \Big)$$

Заметим, что теперь каждое такое слагаемое можно считать за $\mathcal{O}(1)$ с $\mathcal{O}(n)$ предподсчета. Посчитаем префиксные суммы $b_i = a_1 + \ldots + a_i$ в одном цикле как $b_i = b_{i-1} + a_i$. Тогда теперь нам осталось просуммировать слагаемые вида

$$a_j \cdot b_{j-1} \cdot (b_n - b_j)$$

по всем j от 1 до n. Суммарное время работы — $\mathcal{O}(n)$.

Задача K. Магический XML

Заметим несколько свойств, которыми обладает корректная строка обладающая указанными закономерностями:

- Число вхождений символа «<» равно числу вхождений символа «>»
- Число вхождений символа «<» ровно в два раза больше числа вхождений символа «/»
- Для каждого символа латинской строчной буквы, его число вхождений четно (нужно чтобы для каждой буквы была парная ей в закрывающем тэге)
- Букв должно быть хотя бы столько же, сколько символов «<»

Таким образом нужно проверить все эти свойства, если хотя бы одно не выполнено — ответ `Impossible`.

Иначе из имеющихся символов можно конструктивно построить ответ, например так: `<a><b><c><defg...></defg...></c></b></a>`.