Задача A. Просмотр сериала

В первой подзадаче достаточно заметить, что после каждых m серий будет наступать конец дня, поэтому ответ на задачу $\lfloor \frac{n}{m} \rfloor$.

Во второй подзадаче можно промоделировать процесс просмотра всех серий по одному моменту времени (суммарная продолжительность всех серий не превышает 10^6) и для каждого момента проверить, попадает ли он на конец дня.

Впрочем, легче решить сразу полную версию задачи. Для каждой серии найдем момент времени, когда она будет досмотрена до конца — это сумма продолжительностей текущей серии и всех предыдущих серий. Если полученное время делится без остатка на m, то мы получаем удачный день и увеличиваем ответ на 1. Вычислительная сложность такого решения — $O(n)$.

Задача B. Скучный урок

Для решения первой подзадачи достаточно перебрать размеры нарисованного прямоугольника — $a \times b$, где $1 \leqslant a \leqslant n$, $1 \leqslant b \leqslant m$. Для такого прямоугольника количество отрезков будет равно:

$$k(a, b) = a \cdot (b + 1) + b \cdot (a + 1)$$

Если полученное число $k(a, b)$ не меньше t, то увеличиваем ответ. Сложность решения — $O(n \cdot m)$.

Для решения второй подзадачи нужно перебирать только размер стороны a ($1 \leqslant a \leqslant b$). При фиксированном a можно найти бинарным поиском такое число b, что $k(a, b) \geqslant t$ — это можно сделать, поскольку $k(a, b)$ монотонно возрастает при увеличении b. Если найденное минимальное значение b_0 не превосходит m, то к ответу нужно прибавить значение $m - b_0 + 1$. Итого, сложность решения будет $O(n \cdot log(m))$.

Примечание: используя похожие идеи, можно решить данную задачу и в более общем случае: $1 \leqslant n, m, t \leqslant 10^9$.

Задача C. Парные браслеты

Можно заметить, что нам не важен порядок чисел на браслетах, для каждого числа важно только количество, сколько раз оно встречается на первом браслете и на втором браслете.

Для подзадачи 1 достаточно проверить, совпадают ли эти количества для каждого числа. Это легко проверить, например, при помощи сортировки обоих массивов. Если после сортировки массивы стали равными, то ответ 0, иначе — -1.

Для решения следующих подзадач можно заметить, что нас интересуют только числа, которые находятся хотя бы на одном браслете. Во второй подзадаче можно перебрать все возможные пары (a, b) из этих чисел и для каждой промоделировать операцию Боба — поменять a на b и наоборот на одном браслете. После этого проверить, совпадают ли массивы по аналогии с первой подзадачей. Вычислительная сложность — $O(n^3 \cdot log(n))$.

Для следующих подзадач имеет смысл посчитать — сколько существует чисел, что их количества в браслетах отличаются. Если таких чисел нет, то ответ 0. Если таких чисел больше 2, то ответ -1 (за одну операцию не получится изменить количество сразу трёх и более чисел). Если есть всего два таких числа, то, очевидно, нужно применить операцию именно к ним. При этом можно промоделировать операцию Боба или просто проверить, что количество чисел a в первом браслете равно количеству чисел b во втором. Обратите внимание, что не может быть ситуации, когда отличается количество только одного числа, поскольку оба браслета содержат ровно по n чисел.

В зависимости от подзадачи для подсчета количества чисел в массивах можно воспользоваться проходом по массиву (подзадача 3), записать количество чисел для каждого браслета в отдельный массив (подзадача 4) или воспользоваться контейнером, например, *map* в C++ (полное решение). В последнем случае вычислительная сложность решения будет $O(n \cdot log(n))$.

Задача D. Два числа

Подгруппа 1

В этой подгруппе достаточно выполнить то, что дано в условии и можно не обращать внимание на ответ по модулю, так как числа будут гораздо меньше. То есть просто выполняем

$a_i = a_i + a_{i+2}$ и удаляем два последних числа до тех пор, пока не останется два элемента.

Решение $O(n^2)$.

Подгруппа 2

Аналогичное решение с подгруппой 1, только теперь добавляется ответ по модулю, так как числа могут быть большими.

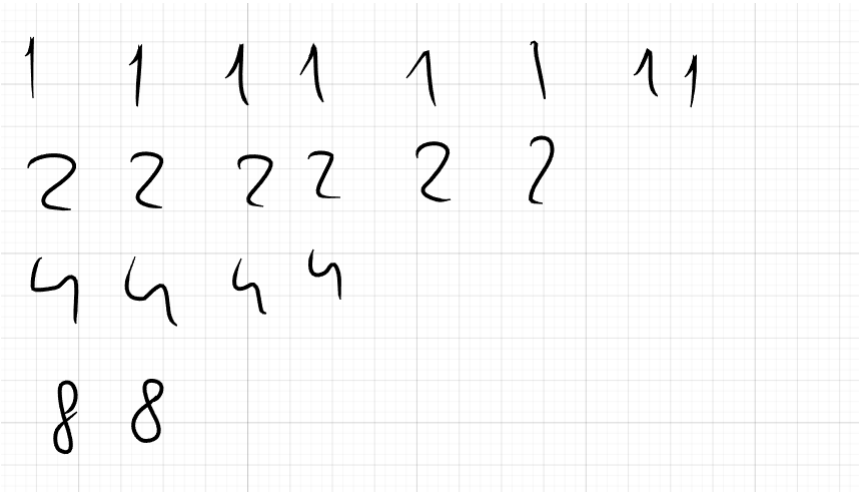
$a_i = (a_i + a_{i+2})\%1000000007$ и удаляем два последних числа до тех пор, пока не останется два элемента.

Решение $O(n^2)$.

Подгруппа 3

Так как все элементы изначально равны 1, то все элементы на следующем шаге будут равны 1 + 1 = 2.

Теперь все элементы равны 2, а значит на следующем шаге будут равны 2 + 2 = и т.д. Получается, что ответом будут числа, равные степени двойки. А степень будет равняться количеству удалений последних чисел, их всего $\frac{n}{2} - 1$.



Решение $O(n)$ или $O(log(n))$ в зависимости от реализации.

Полное решение

Давайте посмотрим, какие элементы участвуют в образовании каждого числа

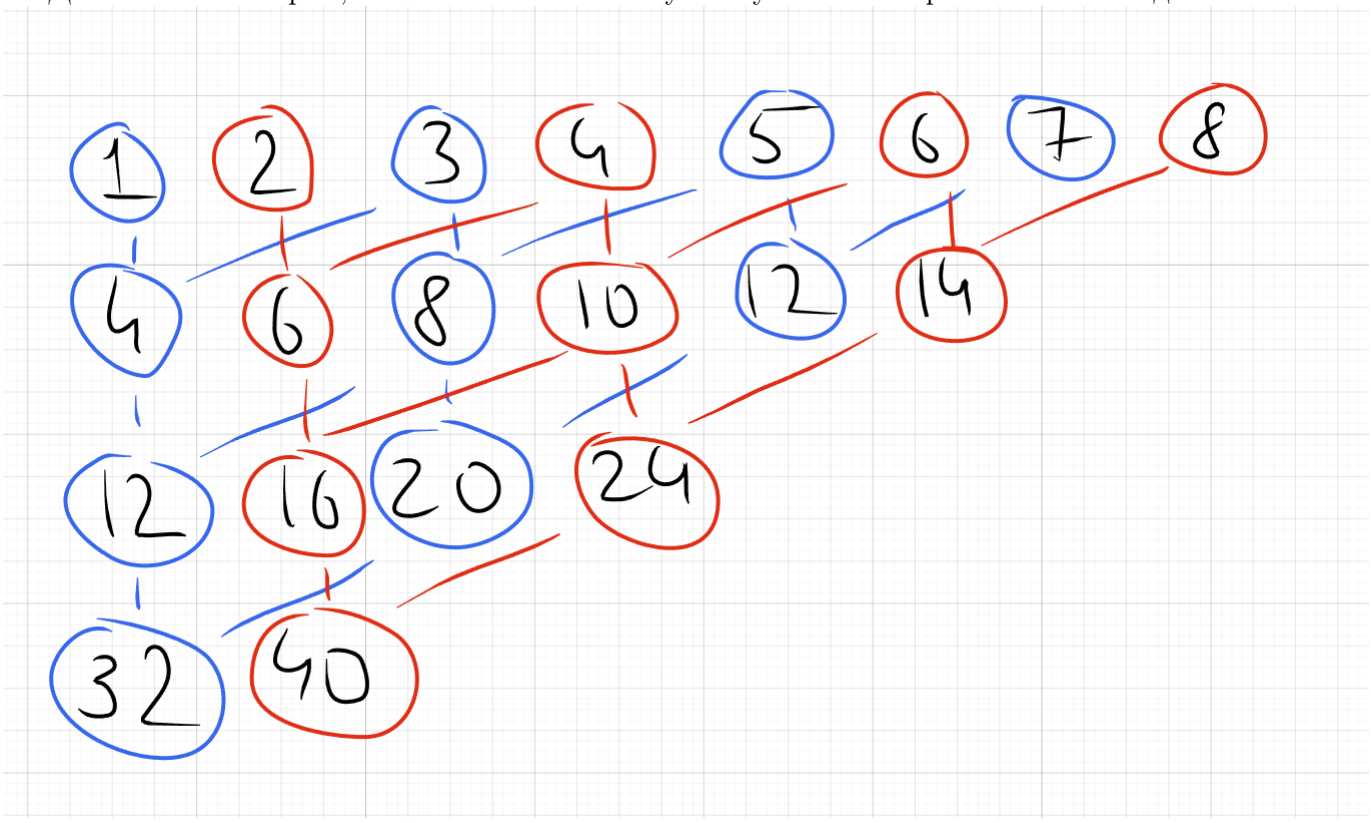


Рисунок наглядно показывает, что два числа получаются независимо друг от друга, то есть в них нет общих элементов.

Так как числа получаются независимо друг от друга, то давайте разобьем элементы на два массива размером $\frac{n}{2}$, где в первый войдут элементы с четным индексом, а во второй — с нечетным. Для каждого массива решение будет одинаковым.

Еще можно заметить, что по сути мы получили два треугольника Паскаля, и нам нужно посчитать — сколько раз каждое число встречается в последнем, а это довольно известная задачка на сочетания.

$ans = ans + v_i \cdot C^i_{n/2-1}$ для всех i от 0 до $\frac{n}{2} - 1$

Так как числа могут быть большими, то здесь еще нужно реализовать модульную арифметику.

Решение $O(n \cdot log(n))$

Задача E. Аварийная дорога

Заметим, что изначально у нас получается дерево. И каждый запрос независим друг от друга.

Подгруппы 1 и 2

Здесь достаточно в лоб заменять ребра и смотреть, какой кратчайший путь получается в новом графе (который может уже не являться деревом), например при помощи алгоритма Дейкстры.

Решение $O(q \cdot n^2)$ для подгруппы 1 или $O(q \cdot n \cdot log(n))$ для подгруппы 2 в зависимости от реализации алгоритма Дейкстры.

Подгруппы 3 и 4

В этих подгруппах можно заметить, что граф всегда будет деревом, где ребра могут менять свое значение.

Давайте запустим DFS от вершины s и в нем посчитаем расстояние $dist_i$ от вершины s до любой другой.

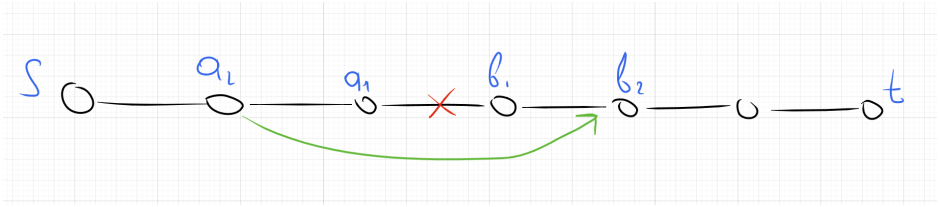
Теперь кратчайшее расстояние между вершинами s и t может измениться только в том случае, если ребро находится на пути между вершинами $ans = dist_t - |dist_{a_i} - dist_{b_i}| + c_i$, иначе $ans = dist_t$.

Принадлежность ребра пути можно каждый раз проверять в лоб, тогда решение $O(q \cdot n)$ для подгруппы 3.

А можно перед тем как обрабатывать запросы — сразу определить все вершины или ребра, которые находятся на пути от s до t. Решение $O(n + q)$ или $O(n + q \cdot log(n))$ для подгруппы 4 в зависимости от реализации.

Подгруппа 5

Здесь дерево — так называемый бамбук.



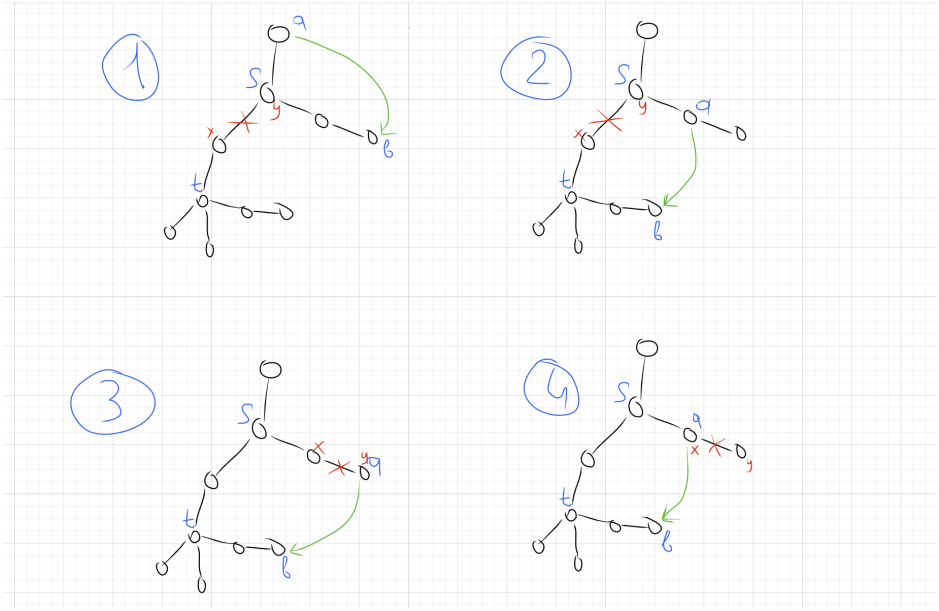
Давайте запустим DFS от вершины s и в нем посчитаем расстояние $dist_i$ от вершины s до любой другой. Пусть $a_1 < b_1$ и $a_2 < b_2$ всегда, если это не так, то просто поменяем их местами.

Необходимо проверить, что если мы удалим ребро a_1b_1 и добавим a_2b_2, то все еще можно достигнуть вершины t.

Это будет верно только тогда, когда $a_2 \leqslant a_1$ и $b_1 \leqslant b_2$. Если это условие выполняется, то ответ $dist_{a_2} + c + dist_t - dist_{b_2}$.

Решение $O(n + q)$.

Полное решение



При удалении ребра мы получаем две компоненты связности. А при добавлении — необходимо проверить, а будут ли вершины s и t в одной компоненте связности.

Давайте запустим DFS от вершины s и в нем посчитаем расстояние $distS_i$ от вершины s до любой другой, а также запомним время входа в вершину и выхода из вершины во время DFS $tinS_i, toutS_i$ — это нужно для того, чтобы можно было узнать, является ли одна вершина предком другой. Вершина a будет предком вершины b тогда, когда $tin_a \leqslant tin_b$ и $tout_b \leqslant tout_a$.

Аналогично запустим DFS от вершины t и посчитаем $distT_i, tinT_i, toutT_i$.

В первом и втором вариантах удаляется ребро на пути между вершинами, значит сейчас s и t в разных компонентах связности. Для того, чтобы ответ был не -1, нужно, чтобы вершины a и b были в разных компонентах связности. Если они в одной компоненте связности, то обе вершины x и y являются предками вершин a и b относительно s или t (поэтому запускали два DFS, чтобы сейчас легко это проверить).

В третьем и четвертом примере удаляется ребро не на пути, и нужно проверить, а может ли получиться еще путь $s - a - b - t$. Нового пути не будет существовать, если хотя бы одна из вершин a или b будет одновременно потомком вершин x и y относительно s или t.

Решение $O(n + q)$.

Задача F. Прохождение уровней

Пусть игрок, начиная с уровня i с x предметами, смог пройти j-й уровень ($j \geqslant i$). Тогда перед j-м уровнем его сила составляла не менее a_j. При этом он уже прошёл $j - i$ уровней, поэтому его сила перед j-м уровнем составляет $x - (j - i)$. Из неравенства $x - (j - i) \geqslant a_j$ получаем, что $x - i \geqslant a_j + j$.

Таким образом, для прохождения j-го уровня необходимо и достаточно выполнить неравенства $x - i \geqslant a_k + k$ для всех $k \in [i, j]$. Значит, если первый непройденный уровень имеет номер r, то r — минимальное, такое, что $i \leqslant r$ и $a_r + r > x - i$. Для решения первой подгруппы достаточно найти подходящее r перебором ($O(n)$ времени на запрос).

Для решения подгрупп 2 и 3 найдём r с помощью структуры данных, эффективно реализующей запрос максимума на отрезке (подойдёт, например, дерево отрезков или разреженные таблицы). Двоичным поиском найдём искомое минимальное r, такое, что $\max\limits_{k \in [i,r]} > x - i$. В зависимости от выбранной структуры данных время обработки запроса составит $O(\log^2(n))$ или $O(\log n)$.

Вместо двоичного поиска можно явно реализовать поиск подходящего r как запрос к структуре данных — например, с помощью спуска по дереву отрезков или двоичных подъёмов в разреженных таблицах. Такое решение также обрабатывает запрос за $O(\log n)$.

Задача G. Три грядки

Заметим, что значения урожайностей участков не меняются, поэтому можно воспользоваться

алгоритмом префикс-сумма для двумерного случая, что позволит находить сумму урожайностей в любом прямоугольнике за $O(1)$.

Теперь для решения первой группы тестов достаточно перебрать ориентацию грядок — $a \times b$ или $b \times a$ (всего 8 вариантов для трёх грядок) и выбрать позиции левого-верхнего угла каждой грядки. Если для данных позиций грядки не пересекаются, обновить ответ суммой урожайностей. Итого, сложность алгоритма $O(n^3 \cdot m^3)$.

Для решения полной задачи надо воспользоваться более продвинутыми идеями. Во-первых, легко проверить, что если 3 прямоугольника не пересекаются, то один из них можно отделить от двух других вертикальной или горизонтальной прямой. Можно считать, что одна грядка отделена **вертикальной** прямой от двух других и находится **справа** от них. Остальные варианты расположения сводятся к этому путём поворота исходного участка на углы, кратные 90 градусов (всего 4 варианта). Далее возможны 2 случая расположения оставшихся двух прямоугольников-грядок: они тоже либо разделены вертикальной прямой, либо горизонтальной прямой. В обоих случаях оптимальное расположение всех трёх грядок можно найти при помощи динамического программирования.

Если все три грядки разделены вертикальными прямыми, то можно посчитать состояния $dp_1[k][y]$ — максимальная сумма, которую можно получить, расположив k грядок ($1 \leqslant k \leqslant 3$), первая из которых начинается в столбце y или правее. Для этого достаточно перебрать расположение левого-верхнего угла грядки в столбце y (n вариантов) и рассмотреть случай перехода в столбец $y + 1$. Итого, все состояния считаются за $O(n \cdot m)$.

Если же две грядки отделены горизонтальной прямой, то для каждой позиции вертикальной прямой (отделяющей третью грядку) нужно уметь находить максимальную сумму, которую можно получить двумя грядками слева от этой прямой. Для этого достаточно посчитать еще два множества значений динамического программирования $dp_2[x][y]$ и $dp_3[x][y]$ — максимальную сумму, которую можно получить одной грядкой, расположенной от $(1, 1)$ до (x, y) и от $(n, 1)$ до (x, y) соответственно. Эти значения считаются похожим образом, как dp_1. Тогда для каждой клетки (x, y) доски мы можем найти оптимальную грядку справа — $dp_1[1][y + 1]$, оптимальную грядку сверху — $dp_2[x][y]$ и оптимальную грядку снизу — $dp_3[x + 1][y]$. Суммой этих трёх чисел нужно обновить ответ. Общая сложность алгоритма — $O(n \cdot m)$.

Так же отметим, что при наличии четырёх и более непересекающихся прямоугольников не всегда один из них можно отделить от других вертикальной или горизонтальной прямой.