,

Задача A. Долго будет Карелия сниться

Треугольник со сторонами a, b, c, где $a \leqslant b \leqslant c$ может существовать только если $a + b > c$, поскольку иначе он вырождается. Потому, если для любого из двух треугольников не выполняется это условие — то ответ сразу 0. Так как числа на вводе необязательно отсортированы, необходимо либо их отсортировать, либо сравнивать максимум с суммой всех остальных сторон за вычетом максимума.

Остаётся проверить, нельзя ли было 2 раза пройтись по одному и тому-же треугольнику. Это можно было сделать как пойдя второй раз в том же направлении, что и в первый, так и в противоположном. Формально это либо $a = d$, $b = e$, $c = f$, либо $a = f$, $b = e$, $c = d$.

Задача B. Сила команды

Нельзя отправить на сборы больше участников, чем k, потому можно сделать $a_i = min(a_i, k)$.

Рассмотрим участника с максимальной силой и участника с минимальной силой. Если разница их сил больше 1, то нам точно не станет хуже, если мы на тех сборах, где был сильный участник, заменим его там на слабого. Потому будем всегда держаться того, что разница сил любых двух участников не превосходит 1.

Давайте отправим на первые сборы a_1 любых участников. Теперь на вторые сборы необходимо сначала отправить тех, кто не попал на первые сборы. Если все такие участники были отправлены, то силы участников опять равны, и потому отправим на оставшиеся места любых участников. Если не удалось на вторых сборах достичь у всех силы один, то пытаемся сделать это на третьих и т.д. Получаем, что ответ на задачу это $\Sigma_{i=1}^{n} \min(a_i, k)$

Задача C. Новый трек

Переформулируем задачу. Пусть $l = \max(1, t-d)$, $r = t+d$. Тогда необходимо посчитать, сколько есть x, таких что:

1. $l \leqslant x \leqslant r$
2. есть не более трёх целых y, таких что $y \geqslant 1$ и y делит x нацело.

Для решения на первую группу тестов достаточно перебрать все числа от l до r, и для каждого из них перебрать делители. Такое решение работает за $\mathcal{O}(d \cdot r)$. Чтобы ускорить решение, заметим, что если число делится на y, то число также делится и на $\frac{x}{y}$, и потому можно перебрать только меньший делитель в этой паре, а он не превосходит $\sqrt{n}$, потому теперь время работы можно оценить как $\mathcal{O}(d \cdot \sqrt{r})$.

Для решения остальных групп удобно использовать следующее наблюдение. Если число имеет два различных простых делителя (p, q), то у него уже есть четыре делителя: 1, p, q, x. Если $x = p^k$ (p простое), то у числа есть $k + 1$ делитель: $p^0, p^1, \ldots, p^k$. Значит, нам подходят только три вида чисел: простые, квадраты простых и единица (у неё нет простых делителей).

В начале решения посчитаем массив простых чисел (pr), не превосходящих $\sqrt{r}$. Быстро сделать это можно с помощью решета Эратосфена. Теперь, чтобы посчитать число квадратов простых от l до r, можно просто перебрать все элементы pr и проверить квадрат каждого из них.

Остаётся посчитать количество простых чисел на отрезке от l до r. В зависимости от реализации этого пункта можно решить группы со второй по четвёртую.

Для решения второй группы будем идти по массиву pr, пока квадрат числа будет не больше x, и если не найдётся ни одного делителя x, то x — простое. Как только нашлось число, на которое делится x, — тут же прерываем цикл и проверяем следующее число.

Для решения третьей группы можно написать решето Эратосфена до числа r, таким образом и проверяя на простоту.

В решении четвёртой группы тестов реализуем решето Эратосфена, но только на числах от l до r. В качестве делителей будем использовать все числа из pr, пусть взяли элемент y, тогда найдём минимальное x, такое что $x \geqslant l$ и x делится на y. Затем будем прибавлять y к x-у до тех пор, пока $x \leqslant r$. Таким образом, мы переберём все числа, которые делятся на y в отрезке от l до r. Решето будем поддерживать на массиве длины $r-l+1$, где i-й индекс массива соответствует числу $x = l+i$.

,

Задача D. Дорожные преобразования

Ключевые пункты и дороги между ними образуют дерево. Количество способов добавить ребро в дерево, создав при этом цикл длины k, это количество пар вершин, расстояние между которыми равно $k-1$.

Решение на первую группу тестов — перебрать все возможные пары вершин и посчитать расстояние между ними с помощью обхода в глубину или в ширину от первой вершины пары до второй. Пар вершин $\mathcal{O}(n^2)$, один обход дерева работает за $\mathcal{O}(n)$, значит время работы решения $\mathcal{O}(n^3)$.

Решение на пятую группу тестов — фиксировать первую вершину пары и запустить от неё один обход, который сразу посчитает, сколько для неё есть вершин на расстоянии $k-1$. Всего n запусков обхода, потому время работы $\mathcal{O}(n^2)$.

Для решения задачи на полный балл воспользуемся подходом сливаемых множеств на дереве. А именно, для начала выберем произвольную вершину, за которую подвесим дерево. Поймём, что расстояние между парой вершин можно выразить как сумму расстояний от каждой из них до наименьшего общего предка (lca). Теперь вместо того, чтобы перебирать пары вершин, будем перебирать их lca. Пусть мы зафиксировали в качестве lca некоторую вершину l и пусть для каждого ребёнка мы храним словарь, ключ которого целое число, обозначающее расстояние от ребёнка до какой то вершины, а значение по этому ключу — количество вершин в поддереве ребёнка на таком расстоянии. Тогда будем добавлять из каждого ребёнка по очереди все значения в словарь для вершины l, при этом одновременно пересчитывая ответ (число пар вершин на расстоянии $k-1$). Если при слиянии словаря ребёнка и словаря вершины l делать *swap*, который работает за $\mathcal{O}(1)$, если в словаре ребёнка больше элементов, чем в родительском, то нетрудно показать, что суммарно будет сделано $\mathcal{O}(n \cdot \log n)$ перекладываний из одного словаря в другой. Говоря простым языком, нужно всегда перекладывать из меньшего словаря в больший, тогда суммарное время работы сильно уменьшится.

Чтобы после добавления значений от всех детей не пробегать по словарю вершины l, прибавляя к каждому ключу единицу, будем поддерживать вместе со словарём для каждой вершины то значение, которое прибавляется ко всем ключам. Чтобы не тратить много памяти, будем очищать те словари, которые больше нет необходимости хранить.

Итоговое время работы решения $\mathcal{O}(n \cdot \log n)$ при реализации на unordered_map (хеш-таблице) и $\mathcal{O}(n \cdot \log^2 n)$ при реализации на map (словарь с сортировкой).

Задача E. Запросы на слонов

Для начала скажем, что $n \leqslant m$. Если это не так, просто будем менять местами n с m и x с y.

Для решения задачи на первую группу можно хранить двумерный массив-доску со всеми выставленными слонами. Для обработки третьего запроса запустим обход в ширину, из клетки можно перейти во все клетки на одной диагонали с ней, их не больше чем $n \cdot 2$. Всего клеток не больше чем $n \cdot m$, потому обход в ширину будет работать за $\mathcal{O}(n^2 \cdot m)$, а всё решение за $\mathcal{O}(q \cdot n^2 \cdot m)$.

Для решения второй группы нужно заметить, что ответ на квадратной доске не больше двух. Чтобы проверять, является ли ответ на запрос нулём, будем поддерживать $set < pair < int, int >>$, для всех позиций, занятых слонами. Чтобы проверять, является ли ответ единицей, нужно поддерживать на каждой диагонали число стоящих там слонов. Диагонали бывают двух типов, потому можно завести две $map < int, int >$. Номер диагонали первого типа для клетки можно получить как $x + y$, а номер диагонали второго типа как $x - y$. Таким образом, ответ может быть один, если при запросе третьего типа на каком то типе диагонали стоит слон (совпадает номер диагонали клетки запроса и номер диагонали клетки слона). Если ответ не ноль и не один, то ответ два (т.к. доска квадратная).

Для решения третьей группы нужно ускорить обход в ширину до $\mathcal{O}(n \cdot m)$. Для этого рассмотрим такой граф: каждая клетка доски это вершина графа, а также каждая диагональ доски это вершина графа. Проведём рёбра из каждой клетки в те диагонали, на которых она лежит (их всего две). Тогда число рёбер в графе оценивается как $\mathcal{O}(n \cdot m)$ и время работы обхода в ширину соответственно тоже.

Рассмотрим решение на полный балл. Как и в решении на вторую группу, будем поддерживать set и две map. Проверить, является ли ответ нулём, можно аналогичным образом. Далее, если $n = 1$, то ответ -1, т.к. слон может на такой доске только стоять на месте. Научимся искать ближайшую

клетку справа по одному типу диагонали. Аналогично можно будет искать ближайшую слева и ближайшие по другому типу диагоналей.

Заметим, что если мы ограничиваем число ходов каким то числом, то множество диагоналей одного типа, которые достижимы, образует непрерывный отрезок. Если ограничить число ходов как c и сравнить с тем, какую диагональ можно достичь, если ограничить как $2 \cdot c$, то можно увидеть, что максимально достижимая диагональ сдвинется вправо на $2 \cdot n - 2$. Произойдёт это потому, что за 2 хода из крайнего положения исходной максимальной диагонали можно пройти в крайнее положение диагонали другого типа и тоже самое сделать ещё раз. Номер диагонали, достижимой за один ход, соответствует диагонали исходной клетки в запросе, максимальная диагональ, достижимая за два хода, получается путём перехода в крайнюю клетку диагонали другого типа из стартовой клетки.

Теперь вспомним, что мы храним все диагонали в map, то есть они отсортированы по номеру. Значит, нам нужно найти ближайшую справа диагональ от стартовой (с помощью lower_bound) и понять, за какое число ходов она достижима. Зная номера достижимых диагоналей за один и за два хода, и что за остальное число ходов нужно прибавить $2 \cdot n - 2$ к числу ходов на два меньше, можно понять, за сколько ходов достижима эта диагональ (формулой или бинарным поиском, что будет дольше). Проделываем это с каждым типом диагоналей слева и справа и берём минимальный из возможных ответов.

Время работы решения $\mathcal{O}(q \cdot \log q)$.