Задача A. Атлеты

Давайте идти в задаче от обратного — возьмем все возможные варианты (которых будет $\binom{n}{k}$) и будем вычитать из них все плохие (те, где победитель заранее определен). Очевидно, что это будет ответом на задачу. Переберем победителя турнира i и для него посчитаем всех его оппонентов j таких, что он точно победит (таких, что $p_j < p_i$ и $s_j < s_i$). Это можно сделать простым линейным проходом, поскольку ограничения в задаче позволяют. Пусть таких атлетов будет cnt. Тогда из ответа надо вычесть $\binom{cnt}{k-1}$ вариантов. Итоговая сложность решения — $O(n^2)$. Биномиальные коэффициенты можно вычислять треугольником Паскаля, то есть, динамическим программированием.

Также эту задачу можно решать и за более быструю асимптотику — можно отсортировать спортсменов по одному из параметров, а по другому хранить какую-то структуру данных, которая умеет делать $+1$ в точке и брать сумму на префиксе (самой простой такой структурой данных может быть дерево Фенвика). Тогда посчитать cnt получится за $O(\log_2(n))$, а биномиальные коэффициенты можно вычислять через факториалы и обратные элементы в кольце по модулю.

Задача B. Доказательство неправоты

Для начала давайте научимся считать количество перестановок для заданного набора пар (i, j), где $a_i < a_j$. Для этого воспользуемся динамическим программированием. Пусть dp_{mask} — количество перестановок, начинающихся с элементов только из множества $mask$. Переберем очередной элемент x. Если x не входит в $mask$, и все y, такие что $a_y < a_x$, принадлежат $mask$, то x можно добавить в перестановку, обновив соответствующее значение динамики.

Чтобы ускорить вычисления, используем массив двоичных масок ls. Где y-й бит числа ls_x выставлен в 1, если $a_y < a_x$. Тогда для проверки возможности перехода достаточно проверить, что $(mask \ \& \ ls_x) = ls_x$.

Таким образом, мы научились считать количество перестановок за $O(n2^n)$.

Тогда для ответа на запросы жюри воспользуемся следующей стратегией: будем выбирать тот знак, который приводит к большему количеству перестановок. После каждого запроса общее количество перестановок будет уменьшаться не более чем в 2 раза. Следовательно, после $\lceil \log_2(n!) \rceil - 1$ запросов останется хотя бы две подходящие перестановки. Это именно то, что нужно.

Осталось научиться восстанавливать любую хорошую перестановку, отличную от перестановки жюри. Это можно сделать разными способами. Например, можно перебрать пару соседних элементов, переставить их местами, а затем проверить, что она соответствует всем сравнениям.

Итоговая асимптотика решения – $O(n \log n \cdot n2^n)$.

Задача C. Кафе для мафии

Будем перебирать столы по порядку начиная с 1 и для каждого стола вычислять, какие мафиози будут сидеть за этим столом. Изначально каждый стол свободен в период времени с 1 до $2n$. Посадим за стол мафиози, который ещё нигде не сидит, с минимальным номером. Пусть номер этого мафиози i. Тогда отрезок свободного времени для стола разбивается на два — с 1 до $a_i - 1$ и с $b_i + 1$ до $2n$. Для каждого нового отрезка времени мы хотим выбрать мафиози, чьё расписание вписывается в отрезок, с минимальным номером. Таким образом, задача сводится к запросу минимума на отрезке среди всех интервалов, входящих в отрезок.

Пусть c — массив размера $2n$, где $[i]$ — отсортированный массив всех пар (b_j, j) таких, что $a_j = i$. Тогда для ответа на запрос (l, r) можно перебрать все $i \in [l, r]$ и для каждого такого i перебирать пары (f, s) из префикса $c[i]$ до тех пока $f \leqslant r$, и затем среди всех перебранных пар взять в качестве ответа минимальный s. Один из переборов можно убрать, заменив отсортированный массив $c[i]$ деревом отрезков для каждого i. Второй из переборов можно убрать, построив дерево отрезков над массивом c. Листья этого дерева отрезков будут содержать деревья отрезков для массивов $c[i]$. Промежуточная вершина k этого дерева отрезков будет содержать дерево отрезков для отсортированного объединения массивов $c[2k + 1]$ и $c[2k + 2]$.

Итоговая асимптотика решения – $\mathcal{O}(n \log^2 n)$. Можно заменить дерево отрезков на корневую декомпозицию, где внутри каждого блока левых концов будут храниться сортированные правые концы. Такое решение будет работать $\mathcal{O}(n\sqrt{n \log n})$.

Задача D. Три подотрезка

Для удобства назовем три непустых непересекающихся отрезка *хорошими*, если в объединение этих трех отрезков каждое число от 1 до m входит одинаковое четное количество раз. Таким образом, в исходной задаче нам нужно посчитать количество хороших троек отрезков.

Для начала научимся эффективно решать упрощенную задачу: задана тройка отрезков $[l_1, r_1]$, $[l_2, r_2]$, $[l_3, r_3]$ — определите, является ли она хорошей. Для этого каждому числу от 2 до m зададим в соответствие случайное число (хеш) по модулю $2 \cdot p$ (далее будем обозначать этот модуль как MOD), где p — простое число размером около 10^{18}. Обозначим хеш, заданный в соответствие числу i как h_i. С числом 1 же поступим хитрее — ему в соответствие зададим число $\frac{MOD}{2} - \sum_{i=2}^{m} h_i$ по модулю MOD. Таким образом, если каждое число от 1 до m встретится одинаковое четное количество раз, то их сумма будет равна 0 по модулю MOD. И теперь, чтобы определять за $O(1)$ является ли тройка отрезков хорошей, нам достаточно предпосчитать префикс-суммы для массива h, а именно массив $pref$, где $pref_i = \sum_{k=0}^{i} h_k$.

Теперь давайте научимся решать изначальную задачу. Для этого рассмотрим тройку отрезков $[l_1, r_1], [l_2, r_2], [l_3, r_3]$. Тогда эта тройка отрезков хорошая, если сумма $(pref[r_1] - pref[l_1 - 1]) + (pref[r_2] - pref[l_2 - 1]) + (pref[r_3] - pref[l_3 - 1])$ равна 0 по модулю MOD. Преобразуем это равенство в другой вид $(pref[l_1 - 1] + pref[l_2 - 1] - pref[r_1]) = (pref[r_2] + pref[r_3] - pref[l_3 - 1])$. Заметим, что в обоих этих суммах по три слагаемых, а так же их всегда можно разделить таким индексом mid, что все элементы первой суммы будут слева от mid (т.е. индексы $l_1 - 1, l_2 - 2, r_1$), а все элементы правой суммы (т.е. индексы $r_2, r_3, l_3 - 1$) будут справа. Зная это, будем решать задачу следующим образом: перебираем этот самый индекс mid от $n - 1$ до 2. Для зафиксированного индекса mid мы будем поддерживать все возможные суммы $(pref[r_2] + pref[r_3] - pref[l_3 - 1])$ для всех индексов $r_2, r_3, l_3 - 1 > mid$ (для этого подойдет структура *unordered_map* в языке C++). И зная все такие суммы, нам остается перебрать все тройки индексов $l_1 - 1, l_2 - 1, r_1 \leqslant mid$ и добавить к ответу количество подсчитанных сум равных $(pref[l_1 - 1] + pref[l_2 - 1] - pref[r_1])$.

При смещении индекса mid влево на 1 нам остается лишь добавить недостающие суммы $(pref[r_2] + pref[r_3] - pref[l_3 - 1])$ в нашу структуру данных.

P.S. Небольшое замечание: чтобы решение проходило по времени необходима одна из двух оптимизаций: зарезервировать размер `unordered_map` каким-нибудь большим числом ($15 \cdot 10^6$ будет достаточно), либо написать рукописную хеш-таблицу.

Задача E. Кратчайший некратчайший путь

Для каждой вершины v графа определим $d_s(v)$ – кратчайшее расстояние от s до v. Эти расстояния можно посчитать с помощью поиска в ширину за $\mathcal{O}(n + m)$. Легко видеть, что (s, t)-путь не кратчайший тогда и только тогда, когда некоторые две соседние вершины v_i и v_{i+1} в нём удовлетворяют $d_s(v_i) + 1 \neq d_s(v_{i+1})$.

Пусть i – минимальный из таких номеров вершин. Тогда $d_s(v_{i-1})+1 = d_s(v_i)$ и $d_s(v_i)+1 \neq d_s(v_{i+1})$. Так как граф неориентированный, $|d_s(v_{i+1}) - d_s(v_i)| \leqslant 1$, а значит $d_s(v_{i+1}) \leqslant d_s(v_i)$. Таким образом, в любом некратчайшем пути всегда есть вершина v_i такая, что $d_s(v_{i-1}) \leqslant d_s(v_i)$ и $d_s(v_{i+1}) \leqslant d_s(v_i)$, а в кратчайших путях таких вершин не бывает.

Решим задачу, зафиксировав вершину v_i внешним циклом. Будем искать из неё два вершинно-непересекающихся пути в $\{s, t\}$, при этом запретим использовать рёбра из v_i в u, где $d_s(u) > d_s(v_i)$. Это осуществимо с помощью поиска потока величины 2 в графе с раздвоенными вершинами за $\mathcal{O}(n + m)$. Истоком в этой сети будет v_i, а в сток (вспомогательную вершину) нужно добавить входящие рёбра из s и t.

Общая временная сложность решения составляет $\mathcal{O}(n(n + m))$.

Задача F. Почти двудольный граф

Для начала решим задачу, если нам дано множество S такое, что граф $G - S$ (G, в котором все вершины из S удалены) – двудольный. Зафиксируем одну из $2^{|S|-1}$ раскрасок (с точностью до перестановки цветов) S в два цвета, назовём её c_S. Теперь будем искать минимальное число рёбер, которые нужно удалить, чтобы граф стал двудольным, при этом раскраска графа в два

цвета согласовывалась с c_S. Конечно же, рёбра с обоими концами в S, которые не согласуются с c_S, нужно удалить вне зависимости от раскраски остальных вершин.

Поскольку $G-S$ двудольный, у него есть его раскраска в два цвета $c_{\overline{S}}$. Раскраска c_S фиксирована, и мы будем пытаться изменить $c_{\overline{S}}$ так, чтобы она согласовывалась с c_S при удалении минимального числа рёбер. Таким образом, наша задача теперь — разбить множество вершин $V(G) \setminus S$ на два класса: те, что надо оставить того же цвета, что и в $c_{\overline{S}}$, и те, что надо перекрасить. При этом нужно учитывать ещё и рёбра между S и $V(G) \setminus S$. Например, если мы решим перекрасить вершину $v \notin S$, то все рёбра вида uv, где $u \in S$ и $c_S(u) \neq c_{\overline{S}}(v)$, нужно будет удалить. Что касается рёбер внутри $G - S$, удалить нужно будет рёбра между вершинами, которые решим перекрасить и вершинами, которые решим не перекрасить.

Мы получаем задачу о минимальном разрезе для нового взвешенного графа: начнём с двух терминальных вершин: t_0 (она соответствует половине, у которой цвет сохраняем) и t_1 (она соответствует половине, которую решим перекрашивать). Затем для каждой вершины $v \in V(G) \setminus S$, создадим её в новом графе. Между t_0 и v создадим ребро веса, равного количеству рёбер между $u \in S$ и v таких, что $c_S(u) \neq c_{\overline{S}}(v)$ (если v решим «отрезать» от t_0, это значит, что мы её перекрасим, то есть потеряем все рёбра между S и v, которые были покрашены правильно). Симметрично, между t_1 и v создадим ребро веса, равного количеству рёбер между $u \in S$ и v таких, что $c_S(u) = c_{\overline{S}}(v)$ (если v решили не перекрашивать, то потеряем ровно эти рёбра). Остаётся добавить в новый граф лишь все рёбра графа $G-S$, они будут веса 1. Согласно рассуждениям выше, минимальный разрез между t_0 и t_1 соответствует минимальному числу рёбер для удаления, чтобы раскраска согласовывалась с c_S (не считая рёбер с обоими концами в S, которые мы сразу же учли).

Заметим, что разрезы веса более k нас не интересуют, поскольку мы хотим удалить не более k рёбер. Таким образом, в графе из n вершин и m рёбер нужный разрез мы найдём за $\mathcal{O}(k \cdot (n+m))$, пустив не более $k+1$ единиц потока в сети. В общем, имея множество S, мы найдём ответ за $\mathcal{O}(2^{|S|} \cdot k \cdot (n+m))$.

Остаётся объяснить, как свести задачу к описанной. Для этого будем постепенно «наращивать» граф, поддерживая ответ (не более k рёбер): начнём с пустого графа без вершин и будем добавлять по одной вершине графа G, то есть, добавляя вершину i, мы добавляем и все рёбра между j и i, которые есть в G при $j < i$. Обозначим граф на первых i вершинах как G_i. G_0 – пустой граф, G_n – это граф G.

Если при переходе от G_i к G_{i+1} ответ X_i (множество рёбер) для G_i подходит и для G_{i+1}, мы его оставляем. Иначе его нужно пересчитать. Для этого построим S, взяв по одному концу у каждого ребра из X_i и, дополнительно, вершину i. Поскольку $G_i - X_i$ – двудольный, то и $G_{i+1} - S$ – двудольный граф. Кроме того, размер S не превосходит $k+1$. Используя процедуру выше, за $\mathcal{O}(2^k \cdot k \cdot (n+m))$ мы сможем найти ответ для G_{i+1} или понять, что он больше, чем $k+1$.

Общее решение, в худшем случае, занимает $\mathcal{O}(2^k \cdot k \cdot n \cdot (n+m))$ времени. Поскольку граф возрастает лишь постепенно, а ответ пересчитывать нужно не всегда, и искать нужно не оптимальный ответ, а любой, не превосходящий k, решение укладывается в лимит по времени.