Задача A. Цивилизация

Для начала избавимся от ситуации, когда могут быть несколько торговых путей, соединяющих одну и ту же пару городов. Несложно видеть, что между каждой такой парой городов достаточно оставить только один торговый путь.

После этого ключевое наблюдение, которое требовалось в этой задаче, состоит в следующем. Цена победы любого игрока равна количеству торговых путей, соединяющих города, не принадлежащих этому игроку.

Докажем это. С одной стороны, несложно видеть, что цена устранения беспорядков не может быть меньше указанной величины. Действительно, пусть есть торговый путь, соединяющий города u и v, и ни город u, ни город v не принадлежат изначально рассматриваемому игроку. Поскольку игрок для победы должен захватить все города, то когда-то он должен будет захватить и город u, и город v. В момент, когда он захватит первый из этих городов, он должен будет потратить одну монету из-за наличия торгового пути из u в v. Поэтому общее количество потраченных монет в процессе захвата мира не может быть меньше, чем количество таких торговых путей.

С другой стороны, несложно видеть, что если новые города будет захватывать только рассматриваемый игрок, а его города другие игроки захватывать не будут, то общее количество потраченных монет будет как раз равно количеству торговых путей, соединяющих города, которые изначально данному игроку не принадлежали.

Таким образом, надо для каждого игрока посчитать количество таких торговых путей. Проведенный «в лоб» такой расчет может быть долгим, но можно поступить следующим образом: для каждого игрока посчитаем количество всех остальных путей, т.е. путей, соединяющих города, из которых хотя бы один изначально принадлежит этому игроку. Если обозначить это количество для i-го игрока как c_i, то цена победы i-го игрока будет равна $M - c_i$.

Рассчитать же значения c_i очень просто: проходим по всем торговым путям; пусть очередной путь соединяет города u и v, и пусть они принадлежат соответственно игрокам a и b. Тогда надо прибавить единицу к числу c_a и, если $a \neq b$, то еще и к числу c_b.

После этого осталось определить максимальный элемент в массиве c и вывести его номер.

Задача B. Деревья на аллее

Пусть отрезок $[L, R]$ — один из оптимальных ответов. Тогда очевидно, что в точке L (как и в точке R) должно быть хотя бы одно дерево. Тогда возможны два случая — либо в точке L есть дерево со стороны аллеи, которой соответствуют числа A и B, либо — со стороны, которой соответствуют числа C и D. Случаи рассматриваются одинаково, рассмотрим оба случая и выберем лучший ответ.

Пусть теперь мы хотим найти оптимальный ответ, при котором в точке L есть дерево с первой стороны аллеи, то есть $L = A + N \cdot B$. Для удобства перенесем начало координат в точку A и найдем соответствующее значение $C' \geqslant 0$ — координату первого дерева на другой стороне аллеи. Тогда с одной стороны аллеи будут расположены деревья в координатах $X = N \cdot B$, а с другой — $X = C' + M * D$. Далее будем искать оптимальный ответ бинарным поиском по длине отрезка. Очевидно, что если для некоторой длины отрезка мы сможем сфотографировать хотя бы K деревьев, то и для большей длины отрезка это получится сделать, поэтому бинарный поиск применим.

Пусть теперь мы зафиксировали длину отрезка W. Нужно проверить, существует ли отрезок длины W, который содержит хотя бы K деревьев. Для этого найдем отрезок длины W, который содержит наибольшее число деревьев. Понятно, что с первой стороны аллеи (где расположены деревья с шагом B) мы 'захватим' фиксированное число деревьев — $\frac{W-1}{B} + 1$ — поскольку отрезок начинается в точке, содержащей дерево на этой стороне. Теперь нужно максимизировать количество деревьев на второй стороне. Поскольку длина отрезка фиксирована, а деревья расположены на одинаковом расстоянии друг от друга, то максимальное число деревьев будет находиться на отрезке, если первое дерево будет расположено как можно ближе к началу отрезка. Как этого добиться? Пусть G — наибольший общий делитель чисел B и D. Тогда при изменении положения начала отрезка, позиция первого дерева на второй стороне аллеи будет изменяться на величину, кратную G. Значит, мы не сможем получить дерево ближе (к началу отрезка), чем $C' mod G$.

Чтобы получить дерево ровно на таком расстоянии нужно сначала решить уравнение $B \cdot X - D \cdot Y = G$. Решение этого уравнения всегда существует и его можно найти при помощи

алгоритма Евклида. Далее, при увеличении позиции начала отрезка на $B \cdot X$, позиция первого дерева на второй стороне уменьшится на G. Тогда, обозначив $H = C'/G$, мы получим, что нужно взять стартовую позицию отрезка $L = B \cdot X \cdot H$. При этом, от произведения $X \cdot H$ нужно взять лишь остаток при делении на D, чтобы не выйти за допустимые границы — в этом случае значение L не будет превышать 10^{18}.

Теперь, зная координату L начала отрезка и длину отрезка W, легко посчитать, сколько деревьев будет содержать этот отрезок и сравнить это число с K.

Задача C. Счастливые моменты времени

Заметим, что в течение одного года может быть не более одного счастливого момента времени. А именно: если последние две цифры года образуют число, большее 0 и не превосходящее 12, то в этот год будет один счастливый момент времени, иначе — ни одного.

Рассмотрим заданный интервал времени. Для начала нужно определить, входят ли в этот интервал счастливые моменты в первый год интервала и в последний год интервала (если соответствующие моменты существуют). Проверить это несложно — необходимо сравнить заданные месяц, день, часы, минуты с последними двумя цифрами года. Отдельно следует рассмотреть случай, когда первый и последний годы интервала совпадают — чтобы не учесть счастливый момент времени два раза.

Далее, каждый год интервала, не являющийся первым или последним, входит в интервал полностью. Значит, если в этот год есть счастливый момент времени, то он так же входит в заданный интервал. Поэтому достаточно посчитать количество лет, которые содержат счастливые моменты времени. Простой способ (для получения 60 баллов) — перебрать все года интервала и каждый проверить отдельно. Это решение работает долго (требуется порядка $2 \cdot 10^9$ проверок). Чтобы его ускорить, достаточно заметить, что среди каждых 100 подряд идущих лет ровно 12 содержат счастливые моменты времени. Поэтому по интервалу можно идти с шагом 100 (пока это возможно), прибавляя каждый раз к ответу 12.

Еще одна полезная идея, которая могла упростить решение — можно отдельно посчитать количество счастливых моментов времени начиная с какого-нибудь очень раннего момента времени (например, ‘00:00 01.01.1000’) до заданных в условии моментов времени, а потом вычесть из одного полученного числа другое. Это позволяет сократить количество особых случаев, в частности, не рассматривать особо случай, когда первый и последний годы интервала совпадают.

Задача D. Магические посохи

На первый взгляд хочется написать какое-то подобие задачи о рюкзаке, но количество воможных посохов слишком велико, и классическое решение задачи о рюкзаке не уложится в ограничение по времени. Тем не менее, достаточно очевидно, что среди посохов одного веса оптимально брать самые длинные, поэтому давайте подумаем, сколько оставить посохов для каждого веса. Несложно видеть, что для веса i необходимо оставить не более W/i самых длинных посохов. Исходя из этого мы получаем, что всего посохов необходимо $\sum_{i=1}^{W} W/i = O(W \log W)$. После этого уже можно написать классическое решение задачи рюкзаке с асимптотикой $O(W^2 \log W)$. Выбор же лучших посохов для каждого веса можно легко реализовать с помощью структуры данных типа «множество» или «куча» за $O(N \log N)$ на то, чтобы поместить изначальные значения для каждого заклинания в структуру данных и $O(W \log W \log N)$ на то, чтобы оставить лучшие посохи.

Задача E. Проектор

Эта задача не должна была представлять особых сложностей, ее можно было решить разными способами, отличающимися по количеству рассматриваемых частных случаев.

По мнению жюри, наиболее простым является следующий подход. Давайте сначала определимся с величиной сдвига экрана. Очевидно, оптимальный вариант — расположить экран так, чтобы его центр совпадал с центром изображения, т.е. сдвинуть экран на расстояние L; это возможно, если $Y \geqslant L$. В противном случае очевидно экран надо двигать в сторону изображения на максимальное возможное расстояние — на Y. Таким образом, в общем случае экран надо сдвинуть на расстояние, равное $S = \min(L, Y)$.

После этого осталось отрегулировать размер изображения. Расстояние между центром экрана и центром изображения получается равно $L - S$ (это всегда положительная величина в силу определения S), соответственно, расстояние от центра изображения до края экрана равно $W/2 - (L - S)$. Соответственно, максимальная ширина изображения, которое может поместиться на экране, в два раза больше и равно $C = W - 2L + 2S$. Осталось сравнить полученную величину C с допустимыми размерами изображения: если $C < A$, то решения нет, если $C > B$, то надо настроить изображение ширины B, иначе ($A \leqslant C \leqslant B$) надо настроить изображение ширины C.

Возможны были и другие подходы, например, стараться выровнять дальний (от центра) край изображения с дальним краем экрана; при аккуратной реализации такое решение также возможно. В подобных решениях надо только не забыть проверить, что и другой (ближний к центру) край изображения попадает в пределы экрана. В изложенном выше решении такой проблемы не возникает, т.к. мы точно знаем, в какую сторону от итогового центра экрана смещено изображение.

Также надо отметить, что в решениях следует избегать использования вещественных чисел, т.к. их точности не будет хватать на тестах, в которых входные данные порядка 10^{18}. Изложенное выше решение работает полностью в целых числах.

Задача F. Divan любит тыквы

Заметим, что запрос a, b можно разбить на две части. Сначала подняться от a до $v = LCA(a, b)$ и потом спуститься до b.

Давайте научимся считать ответ на часть запроса - подняться до v. Посмотрим на то, какой следующей тыквой можно удивить друзей. Это самый глубокий наш предок такой, что тыквы в нем больше нашей тыквы. Можно предподсчитать этого предка. Для этого построим бинарные подъемы, храня не только вершину, в которую мы придем, но и максимум на пути. Таким образом, за $O(n \cdot \log(n))$ мы построим их и для каждой вершины найдем на сколько максимально можно подняться по тыквам, не большим нашей.

Мы получили некоторое новое дерево, где в предке тыква всегда больше, чем в нас. Давайте построим на нем бинарные подъемы. Теперь, с их помощью, мы можем подниматься до вершины, не выше чем v и считать сколько раз друзья удивятся и какая тыква будет последней.

Теперь надо научиться спускаться. Для этого попробуем сначала упростить задачу. Если бы это был бамбук, то можно было бы посчитать точно также ближайшую снизу тыкву больше нашей.

Отлично, тогда, можно рассмотреть все пути от корня. Все они являются бамбуками. К сожалению, быстро построить бинарные подъемы уже не получится.

Давайте попробуем решить задачу с некоторыми изменениями: пусть к бамбуку можно подвешивать вершину или удалять самую нижнюю вершину. Давайте будем для каждой вершины поддерживать ответ, если бы мы пошли из нее вниз. Тогда, заметим следующий факт: если сверху вниз выписать последнюю взятую тыкву, то получится невозрастающий массив. Это доказывается достаточно просто: пусть это не так. Тогда есть такая вершина u, что пойдя из нее вниз последней мы возьмем какую-то тыкву x_u. При это для $pre[u]$ это будет $x_{pre[u]} < x_u$. Но где-то ниже u есть x_u, которая точно удивит друзей, а значит мы бы ее точно взяли.

Пользуясь этим замечанием, мы можем понять, что добавление новой тыквы снизу бамбука добавит ее в пути некоторого суффикса. Давайте поддерживать два дерева отрезков. Одно для количества удивлений друзей, другое для последней взятой тыквы. Тогда спуском по второму дереву мы найдем какой суффикс изменит новая тыква. На нем прибавим 1 и приравняем новой тыкве последнюю взятую.

Теперь посмотрим, как можно удалить последнюю тыкву. Понятно, что можно просто откатить изменения. Для этого воспользуемся идеей персистентности.

Теперь мы можем запустить обход графа в глубину и для каждой вершины сохранить версию дерева отрезков для пути от корня в нее. Это займет у нас $O(n \cdot \log(n))$ времени.

Вернемся к исходной задаче. Мы поднялись в $v = LCA(a, b)$ и последняя взятая тыква какая-то x. Если это тыква из v, то мы можем просто посмотреть в версию дерева отрезков для b и сразу получить ответ. Однако, не всегда все так хорошо. Нам нужно найти первую тыкву большую x на пути $v - b$. Для этого построим еще одно персистентное дерево отрезков, на этот раз просто на максимум. Теперь, задача свелась к спуску на суффиксе.

Таким образом поиск ответа происходит в три фазы.

- Подняться до $v = LCA(a, b)$, используя бинарные подъемы на дереве больших предков
- Найти следующую интересную тыкву на пути $v - b$, используя спуск по дереву максимумов.
- Взять ответ для нее дереве отрезков для b.