Задача 1. Кирпичная стена

Нетрудно понять, когда стену построить не получится: либо когда всего в стене два ряда и в них разное число кирпичей, либо когда число кирпичей в стене изначально нечётное число, а число рядов чётно. Оказывается в остальных случаях всегда можно построить стену.

Пусть в самом низком ряду A кирпичей, а в самом высоком ряду B кирпичей. Зафиксируем высоту $H \geqslant B$, до которой мы хотим достроить стену. Пусть S – число кирпичей, которых не хватает до такой стены высоты H. Понятно, что оно должно быть чётным. Тогда необходимым условием построения стены высоты H является неравенство $H - A \leqslant S - (H - A)$. Действительно, поскольку в самый низкий ряд нужно доложить $H - A$ кирпичей, при этом каждый раз мы должны класть один кирпич ещё куда-либо. Оказывается, что это будет и достаточным условием (при чётном S). Доказать это можно индукцией. Будем доказывать, что это неравенство для ряда наименьшей высоты будет сохраняться, если мы каждый раз будем класть по кирпичу в два ряда наименьшей высоты. Базой индукции можно считать случай когда остается доложить всего 2 кирпича. Из неравенства $H - A \leqslant S - (H - A)$ следует, что кирпичей не хватает в двух разных рядах, понятно, что они и будут как раз иметь наименьшую высоту и за один раз мы достроим стену.

Пусть теперь мы положили по кирпичу в два наименьших ряда, докажем, что неравенство $H - A \leqslant S - (H - A)$ сохранилось. Если наименьший ряд после добавления кирпича остался наименьшим, то неравенство сохранилось, поскольку числа с обеих сторон от неравенства уменьшились на 1. Наименьший ряд мог перестать быть наименьшим, только если имелось три ряда высоты A, два из которых мы увеличили на 1. Тогда нужно проверить, что выполнено неравенство $H - A \leqslant S - 2 - (H - A)$. Но поскольку в двух рядах ещё не хватает по $H - (A - 1)$ кирпичей, то $S - 2 - (H - A) \geqslant H - (A + 1) + H - (A + 1)$. Заметим, что $H - (A + 1) + H - (A + 1) \geqslant H - A$ при $H \geqslant A + 2$, поэтому осталось рассмотреть случай $H = A + 1$. Тогда надо проверить условие $S - 1 \geqslant 1$, которое является верным, поскольку S – чётное число. Тем самым утверждение доказано.

Обозначим через D число кирпичей в исходной стене. Таким образом из предыдущего рассуждения следует, что нужно найти наименьшее H такое, что $H \geqslant B$ и $H - A \leqslant H \cdot N - D$, при этом число $H \cdot N - D$ должно быть чётным. Нетрудно из этих условий найти H явным образом.

Заметим, что из доказательства решения также следует, что в оптимальном варианте можно всегда класть кирпичи в два наименьших по высоте ряда. Отсюда получаем различные медленные решения, например, когда происходит непосредственная эмуляция таких ходов, либо с некоторой оптимизаций – в наименьшие ряды можно сразу добавлять по нескольку кирпичей, пока они остаются двумя наименьшими.

Задача 2. Хитрый автомат

Пусть мы знаем, что на местах i_1, i_2, i_3, i_4 стоят ненулевые элементы. Ограничения гарантируют, что такие 4 индекса всегда найдутся.

Отправим интерактору 4 запроса:

? i_2 i_3 i_4,
? i_1 i_3 i_4,
? i_1 i_2 i_4,
? i_1 i_2 i_3.

В ответ мы получим числа:

$$\begin{cases} a = k_{i_2} \cdot k_{i_3} \cdot k_{i_4} \\ b = k_{i_1} \cdot k_{i_3} \cdot k_{i_4} \\ c = k_{i_1} \cdot k_{i_2} \cdot k_{i_4} \\ d = k_{i_1} \cdot k_{i_2} \cdot k_{i_3} \end{cases}$$

Получили систему уравнений, из которой можно однозначно найти числа k_{i_1}, k_{i_2}, k_{i_3}, k_{i_4}:

$$\begin{cases} k_{i_1} = \sqrt[3]{\frac{bcd}{a^2}} \\ k_{i_2} = \sqrt[3]{\frac{acd}{b^2}} \\ k_{i_3} = \sqrt[3]{\frac{abd}{c^2}} \\ k_{i_4} = \sqrt[3]{\frac{abc}{d^2}} \end{cases}$$

Таким образом, за 4 запроса мы нашли 4 числа. Каждое следующее неизвестное число можно находить за 1 запрос: отправляем индексы двух уже известных ненулевых элементов, а также индекс неизвестного элемента, ответ интерактора делим на произведение известных элементов и получаем искомое число.

То есть, зная индексы 4 ненулевых элементов, можно решить задачу за N запросов. Дополнительные запросы во второй и третьей подзадаче были даны, чтобы найти эту четверку. Сделать это можно было, отправляя интерактору случайные тройки индексов (избегая повторений) до тех пор, пока не получим ответ, отличный от нуля. Генерировать тройки можно было, например, следующим образом: создать массив из N элеметов, заполненный нулями, и на первые три места поставить единицы. Затем вызвать функцию std::shuffle, которая случайным образом перемешает массив, и посмотреть, на каких местах оказались единицы. Это и будет очередная случайная тройка. После того, как ненулевая тройка найдена, нужно найти четвертый ненулевой элемент. Здесь уже можно обойтись обычным генератором случайных чисел от 1 до N (так же избегая повторений).

Если количество нулевых элементов равно $2/3$ от общего количества элементов, то вероятность найти ненулевую четверку за 200 запросов $\approx 98\%$. Если вам вдруг не повезло с первого раза, можно было сделать еще одну попытку.

Задача 3. Иннокентий и путешествие

Ограничения первой и второй групп достаточно небольшие, чтобы можно было просто выполнить все запросы.

Решения следующих групп можно упростить, если повернуть плоскость на 45 градусов, выполнив следующее преобразование: $(x, y) \to (x + y, x - y)$. Также можно заметить, что если на расстоянии r находятся хотя бы k интересных мест, то на расстоянии $r + 1$ так же будут хотя бы k мест, а значит, будем искать минимальное расстояние с помощью бинарного поиска.

Для третьей группы нужно заметить, что ограничения на координаты достаточно небольшие, чтобы можно было использовать двумерный массив префиксных сумм: в массиве на позиции (x, y) будем хранить количество мест, для которых координаты меньше или равны x и y соответственно. Тогда с помощью формулы включений-исключений можно отвечать на запрос за $O(\log(a_{max})))$, где a_max - наибольшая координата.

Для того чтобы решить последнюю группу, нужно построить дерево отрезков по x-координате, в каждом узле которого хранить y-координаты, расположенные в порядке возрастания. Тогда на запрос можно отвечать за $O(\log(a_{max}) \log^2(n))$, если с помощью бинарного поиска искать количество нужных точек в нужном диапазоне.

Задача 4. Шпион в Берляндии

Построим граф, вершинами которого являются города, а рёбрами — участки дороги. Так как из каждой вершины в любую другую существует ровно один путь, не проходящий дважды по одному ребру, то граф является деревом. Найдём кратчайший путь из s в t. Назовём этот путь $P = su_1u_2 \dots u_kt$. Рассмотрим любое ребро на пути P. Это ребро — мост между двумя частями графа, в одной из которых содержится s, а в другой t. Проходя по ребру, мы переходим в другую часть графа. Тогда, раз мы начали обход в части графа, где содержится s, а закончим его в части графа, где содержится t, то такое ребро необходимо будет пройти нечётное число раз. Любое же ребро, не лежащее на пути P, разбивает граф на часть, в которой мы должны начать и закончить (где есть и s, и t), и оставшуюся часть. Такое ребро надо пройти чётное число раз. Теперь в i-м ребре можно

понизить значение c_i до ближайшего числа нужной чётности c'_i, в зависимости от того, лежит ли ребро на пути P. Заметим, что можно удалить из графа все рёбра i, для которых c'_i равно нулю: проходить по ним нельзя. (Конечно, для подзадачи 2 об этом можно не думать.) Граф разобьётся на компоненты связности. Понятно, что P будет лежать в одной из них. Все же компоненты связности графа недоступные из вершин на пути P выкинем и забудем про них навсегда. Заметим, что явно на компоненты связности разбивать не нужно, нужно лишь правильно учитывать c'_i в дальнейшем решении.

Можно построить обход, который посещает рёбра на пути P ровно один раз, а рёбра вне пути P (не считая выкинутых) ровно 2 раза. (Этого достаточно для решения подзадачи 2.) Для этого запустим обход в глубину из вершины s, но не будем в нём идти по ребру su_1. (Без умения писать обход в глубину можно было решить подзадачу 1, в которой граф является простым путём.) После этого обхода пройдём по ребру su_1 и снова запустим обход в глубину, уже из вершины u_1, опять же не разрешая ему ходить по рёбрам на пути P. Затем пройдём по ребру u_1u_2. Продолжим делать такие обходы из всех вершин пути P, сделав последний обход в вершине t. (При обходах в глубину нам нужно понимать, какие ребра находятся на пути P, а какие нет. Если проверять каждое ребро на ходу, решим только подзадачу 3. Эффективнее найти путь P заранее.)

Теперь изменим наш обход так, чтобы по i-му ребру пройти ровно c'_i раз. Заметим, что по каждому ребру нам осталось пройти ещё чётное число раз, так как c'_i чётное для рёбер вне P и нечётное для рёбер на P. Это сделать нетрудно: в последний раз проходя по ребру i, пройдем по нему в обратном направлении, а затем снова в прямом такое количество раз, чтобы суммарное количество проходов осталось c'_i, а затем только продолжим обход. Так мы расширили путь найденный ранее до максимально длинного и решили подзадачу 4.

Задача 5. Предсказать игру

Состояние ГПСЧ в задаче задаётся остатком от деления по модулю M, то есть числом в диапазоне от 0 до $M-1$. В задаче дана функция перехода от текущего состояния к следующему, которая имеет вид: $f(x) = (a \cdot x + b) mod M$.

`Подзадачу 1` можно решить каким-нибудь перебором.

Для начала научимся отвечать на вопросы первого типа. Для этого нужно вычислить $f^k(x)$. Если просто k раз применить функцию f к состоянию, то будет слишком медленно, т.к. k может быть до 10^9. Заметим, что для любых двух функций вида $x \to (a \cdot x + b) mod M$ можно легко вычислить их композицию, причём она будет записываться в таком же виде. Благодаря этому можно сначала вычислить функцию f^k (т.е. функцию перехода на k шагов вперёд) с помощью быстрого возведения в степень за $O(log\ k)$, а потом просто применить её к заданному x.

Этого достаточно, чтобы решить `подзадачу 4`. Представим себе ориентированный граф с M вершинами и M рёбрами. Каждое возможное состояние ГПСЧ — это вершина графа x, из которой выходит ровно одно ребро, ведущее в вершину $f(x)$, соответствующую следующему состоянию. Будем полагать, что M — простое число (это верно всюду кроме последней подзадачи). Тогда можно заметить, что граф обязательно является набором отдельных циклов. Чтобы это доказать, можно например от противного доказать, что при $x \neq y$ не может получиться $f(x) = f(y)$. При желании можно даже доказать, что всегда один цикл состоит из одной вершины, а все остальные циклы имеют одинаковую длину — но это уже сложнее.

Теперь можно решить `подзадачу 2`, научившись отвечать на вопросы второго типа за $O(sqrt(M))$. В этих вопросах требуется найти минимальное k, такое что $f^k(x) = y$ для заданных x и y (по сути это обратная задача). Для решения можно использовать известный алгоритм giant-step baby-step, который является естественным применением методов корневой декомпозиции или meet-in-the-middle. Выберем число $B = O(sqrt(M))$. Вычислим последовательность «ключевых» состояний от x, каждый раз шагая на B шагов вперёд: $x, f^B(x), f^{2B}(x), f^{3B}(x), f^{4B}(x), ..., f^{qB}(x)$.

Здесь $q = M/B = O(sqrt(M))$ — общее количество ключевых состояний. Эти ключевые состояния покрывают цикл элемента x достаточно плотно, так что в любой непокрытой цепочке получается меньше B вершин. Значит если мы вычислим вторую последовательность, шагая от y по одному шагу: $y, f(y), f^2(y), f^3(y), f^4(y), ...f^{B-1}(y)$, то в ней непременно будет хотя бы одно ключевое состояние. Наша задача — найти совпадающий элемент этих двух последовательностей, то есть индексы i

и j, такие что: $f^{iB}(x) = f^j(y) \Rightarrow f^{iB-j}(x) = y$. Таким образом получается искомый шаг $k = i \cdot B - j$. Если совпадение не нашлось, значит x и y точно лежат в разных циклах. В каждой из двух последовательностей по $O(sqrt(M))$ элементов, так что совпадения можно найти сортировкой или деревом поиска за $O(sqrt(M)logM)$, или хеш-таблицей за $O(sqrt(M))$ в среднем.

Описанное выше решение работает за общее время $O(Qsqrt(M))$.

Чтобы решить `подзадачу 3`, нужно ускорить решение до $O(sqrt(Q \cdot M))$. Для этого заметим, что нам необязательно для каждого нового вопроса заново размечать циклы графа ключевыми элементами. Можно заранее на каждом цикле длины L выбрать произвольный элемент и посчитать от него ключевые состояния, каждый раз прыгая на B шагов вперёд. Получается всего $O(L/B)$ элементов в одном цикле, что в сумме по всем циклам даёт $O(M/B)$ ключевых точек. Все эти ключевые точки вычислим заранее один раз и занесём в хеш-таблицу, причём для каждой точки запомним для номер цикла и её «позицию» в нём. Тогда для ответа на вопрос для чисел x и y надо из обоих элементов прошагать вперёд до первой попавшейся ключевой точки — это произойдёт за $O(B)$ шагов в худшем случае. По ключевым точкам можно определить положения чисел x и y в цикле и, соответственно, расстояние между ними. Суммарно получается время работы $O(M/B + QB)$. При выборе $B = sqrt(M/Q)$ это даёт общую асимптотику $O(sqrt(M \cdot Q))$.

Заметим, что в этом случае мы делаем больше работы на предподсчёте, за счёт чего мы можем отвечать быстрее на вопросы. Этот подход напоминает метод rainbow tables, с помощью которого выполняют некоторые атаки на криптографические хеши вроде MD5.

Касательно реализации этого алгоритма: следует размечать циклы ленивым образом: строить ключевые точки на цикле лишь тогда, когда впервые в вопросе попадается состояние из него.

Дополнительная сложность заключается в том, что структуры данных STL очень медленные, и для решения подзадачи 3 целиком необходимо написать хеш-таблицы вручную.

В последней `подзадаче 5` модуль M может быть составным числом. Это сильно изменяет структуру графа: он уже не всегда состоит только из циклов, к каким-то вершинам циклов могут быть прицеплены входящие деревья. Однако можно вывести из соображений теории чисел, что если начать шагать вперёд от любого состояния, то за 30 шагов обязательно попадёшь на какой-нибудь цикл. Худший случай получается например при: $M = 2^{29}$, $a = 2, b = 0, x = 1$ — здесь требуется 29 шагов, чтобы прийти в одноэлементный цикл $0 \to 0$.

Таким образом, можно адаптировать предыдущее решение к имеющемуся случаю. Для этого при получении состояний x и y нужно сначала сделать 30 шагов вперёд, а уже потом делать всё так же, как раньше — шагать до ключевой точки и/или размечать ключевыми точками цикл. Найдя таким образом предполагаемое положение x и y на цикле и определив расстояние d, следует проверить быстрым возведением в степень, действительно ли $f^d(x) = y$. Если нет, тогда нельзя попасть из x в y, т.к. y лежит не на самом цикле, а в одном из входящих деревьев.