Колода для фокусов

Автор задачи и разработчик: Даниил Орешников

Для решения **первой и третьей подзадач** можно было просто просимулировать описанный процесс. При $k \leqslant 2$ в колоде находится всего $n \leqslant 64^2$ карты, и ограничения этой подзадачи позволяли для каждого запроса заново запускать симуляцию из стартового состояния. Понятно, что за n перемешиваний положение любой карты гарантированно зациклится, потому что есть только n возможных мест для карты, поэтому первую подзадачу можно было пройти решением за время $\mathcal{O}(n^2 m)$.

Если останавливать симуляцию после первого зацикливания, то можно пройти и третью подзадачу. Как мы докажем дальше, вся колода возвращается в исходное состояние через ровно $\log_2 n$ перемешиваний, поэтому такое решение с отсечением будет работать за время $\mathcal{O}(nm \log_2 n)$.

Во **второй подзадаче** тоже достаточно было запустить симуляцию, пользуясь тем, что все x_i равны, то есть мы следим за перемещениями одной карточки по колоде. Это позволяет не хранить всю колоду целиком, а хранить только текущее положение интересующей нас карточки. Если мы знаем номер ее изначальной позиции, дальше провести симуляцию несложно: если карточка находится на позиции $i \leqslant \frac{n}{2}$, то после перемешивания она окажется на позиции $2i - 1$; если же она находится во второй половине, то ее номер станет равен $2i - n$.

Научимся искать номер карты со значением x в колоде. Заметим, что на самом деле `base64`-строка — это запись номера в системе счисления с основанием 64. Переведем этот номер в привычный численный вид, заменим каждый символ на шестибитную двоичную запись его позиции в алфавите: '`$`' заменится на '000001_2', а '`a`' — на 38, то есть '100110_2'. Теперь, если сконкатенировать двоичные записи всех символов значения карты, получится битовая строка длины $6k \leqslant 60$, которая может быть переведена в число типа `long long` или `Long`.

Здесь еще надо не забыть, что в условии нумерация позиций в колоде начинается с 1, а полученные нами числа имеют значения от 0 до $2^{6k} - 1$.

Получаем решение второй подзадачи: описанным образом переведем все x_i и y_i в порядковые номера, положим все номера y_i в `map` и запустим симуляцию для x, которая за $\mathcal{O}(1)$ будет вычислять новую позицию карты описанным выше образом, и отвечать на все запомненные запросы про новую позицию x в колоде.

Для решения **четвертой подзадачи** было достаточно воспользоваться фактом, что перемешивание колоды имеет период $\log_2 n$, который можно было, если не доказать формально, то заметить, запустив симуляции на небольших k. Применим решение третьей подзадачи, используя длинную арифметику. Для решения необходимы простейшие операции вроде умножения на 2 и вычитания степени двойки ($n = 2^{6k}$).

Для каждого запроса переведем x_i и y_i в численный вид, и проведем симуляцию для конкретного x_i, пока не попадем в y_i или не зациклимся. Такая симуляция займет $\log_2 n = 6k$ действий с длинной арифметикой, и итоговое решение будет работать за время $\mathcal{O}(m \cdot (6k)^2)$. При достаточно аккуратной реализации это решение также проходило и **пятую подзадачу**.

Во всех следующих подзадачах будем считать, что номера позиций в колоде начинаются с нуля. Заметим следующий факт: операция перемешивания колоды — это просто циклический сдвиг двоичной записи позиции карты на 1 влево. Действительно, если переписать формулы для изменения позиции карты из предыдущей подзадачи в 0-нумерации, то

$$\texttt{next}(i) = \begin{cases} 2i & \text{если } i < \frac{n}{2} \\ 2i - n + 1 & \text{иначе} \end{cases}$$

Посмотрим, что на самом деле делают эти формулы. Если у числа i ноль в старшем бите, оно просто умножается на 2. Если же старший бит единичный, то число умножается на 2, старший бит стирается, и выставляется самый младший. Это и есть в точности описание циклического битового сдвига на 1 влево. Получаем ключевое утверждение задачи: *если карточка x оказалась на позиции y спустя t перемешиваний, y является циклическим битовым сдвигом x на t влево.*

Пользуясь доказанным фактом, можно решить **четвертую подзадачу** другим способом: для каждого запроса независимо перевести x_i и y_i в численный вид, а затем линейным перебором величины сдвига найти минимальную величину циклического сдвига, позволяющую получить один номер из другого. Асимптотика времени работы такого решения равна $\emptyset(m \cdot (6k)^2)$.

Для решения **пятой и шестой подзадач** тем же способом можно было воспользоваться структурой `bitset` или хранить номера в виде массивов с элементами типа `long long` или `Long`. Это позволяет сократить сравнение двух длинных чисел в константу (64) раз, что гарантированно позволяло пройти пятую подзадачу, и при аккуратной реализации — шестую.

Ограничение в $k = 1024$ позволяло проще сравнивать два длинных числа, так как $\log_2 n = 6k$ делится на 64, а значит каждое число состоит из целого количества `long long`, что делает сравнение «на месте» без явного вычисления сдвига более простым в реализации.

Наконец, чтобы получить решение за время $\mathcal{O}(mk)$, достаточно воспользоваться алгоритмом Кнута-Морриса-Пратта, посчитав префикс-функцию на битовой строке, полученной как конкатенация x_i, специального символа и дважды повторенного y_i. Поскольку в тандемном повторе y_i содержится любой циклический сдвиг y_i как подстрока, можно будет за линейное от длины строки время найти все подходящие величины циклического сдвига и вывести минимальную.