Задача A. Сложности с жетоном

Посмотрим на операцию удаления второго символа. Заметим, что если после нее идет операция удаления первого символа, то это равносильно удалению первого символа дважды подряд. Сделаем соответствующую замену, пока она возможна.

Если финальная строка имеет длину (k) больше 3, то операции над последними двумя символами и первыми двумя символами не влияют друг на друга. Поэтому если рассмотреть результат замен операций, описанных выше, после операции удаления второго символа из начала будет удаляться только второй символ, а первый меняться не будет. Аналогично с последним и предпоследним. Таким образом, конечная строка будет представляться конкатенацией `s[l]`, `s[i..j]` и `s[r]`, где $l < i \leqslant j < r$. Заметим теперь, что на самом деле то же самое верно и для $k = 3$, а для $k < 3$ ответ будет просто минимальной последовательностью из k символов исходной строки, что можно получить линейным проходом по строке.

Если мы хотим получить минимальную лексикографически конкатенацию трех описанных подстрок, мы хотим минимизировать `s[l]`, среди всех таких выбрать такое l, для которого `s[i..j]` минимально, а среди всех таких равных выбрать такое, для которого минимальное `s[r]` минимально.

Заметим, что чем меньше l, тем меньше можно будет выбрать `s[i..j]`, так как оно ограничено только условием $i > l$. Таким образом, l равно первой позиции минимальной буквы в строке. Найдем i и j. По тем же самым соображениям, мы хотим найти самую левую позицию в строке минимальной подстроки длины $j - i + 1 = k - 2$.

Допишем к строке в конец граничный символ и построим *суффиксный массив* и `lcp`, равное максимальному общему префиксу двух соседних суффиксов в суффиксном массиве. Это делается *алгоритмом Касаи* за укладывающееся в ограничения время. Теперь, найдем минимальный суффикс среди всех суффиксов после l-го символа строки. Пройдемся по следующим в лексикографическом порядке суффиксам до тех пор, пока значение `lcp` больше или равно $k - 2$, то есть пока подстрока нужной длины так же минимальна. Выберем из всех подходящих самую левую, получим наши i и $j = i + k - 3$.

Осталось только пройтись по всем $r > j$ и выбрать минимальный. Ответ будет конкатенацией `s[l]`, `s[i..j]` и `s[r]`. Асимптотика решения — $\mathcal{O}(n \log n)$. Альтернативно можно было искать минимальную подстроку заданной длины с помощью хэшей.

Задача B. Квадраты Фибоначчи

Оказывается, что

$$\sum_{i=0}^{n} f_i^2 = f_n \cdot f_{n+1}$$

Докажем это по индукции. База: $n = 0$

$$\sum_{i=0}^{0} f_i^2 = 1 = f_0 \cdot f_1$$

Переход: знаем, что

$$\sum_{i=0}^{n} f_i^2 = f_n \cdot f_{n+1}$$

Прибавим к обоим частям по f_{n+1}^2

$$\sum_{i=0}^{n+1} f_i^2 = f_n \cdot f_{n+1} + f_{n+1} \cdot f_{n+1}$$

$$\sum_{i=0}^{n+1} f_i^2 = f_{n+1} \cdot (f_n + f_{n+1})$$

$$\sum_{i=0}^{n+1} f_i^2 = f_{n+1} \cdot f_{n+2}$$

Осталось научиться вычислять n-е число Фибоначчи по модулю. Это достаточно стандартная задача на оптимизацию динамического программирования возведением матрицы в степень. Подробнее можно почитать, например, здесь: `https://e-maxx.ru/algo/fibonacci_numbers`.

Задача C. Похожие заказы

Есть несколько разных способов решения данной задачи. Вот пример двух:

1. Заметим, что одну строку можно преобразовать в другую применением шифра Цезаря тогда и только тогда, когда разности кодов соседних символов у обеих строк совпадают. Формально, строка s преобразуется в строку t тогда и только тогда, когда для любого i выполняется условие $s_i - s_{i+1} = t_i - t_{i+1}$. Таким образом можно построить массивы $ds_i = s_i - s_{i+1}$ и $dt_i = t_i - t_{i+1}$, а затем проверить что ds является циклическим сдвигом dt. Что можно сделать за $O(n)$ с использованием хэшей, Z-функции или префикс функции.

2. Либо можно было заметить, что всего бывает $|25 - (-25) + 1| = 51$ различных значений параметра шифра Цезаря d. Можно перебрать это значение, применить преобразование, а затем аналогично предыдущему способу проверить является ли одна строка циклическим сдвигом другой.

Задача D. Быстрый перевод

Базовое решение — пройтись по всем i от 59 до 0 в убывающем порядке, пытаясь перевести 2^i. Поскольку по условию $x \leqslant 10^{18} < 2^{60}$, то этот алгоритм будет просто по очереди заменять в двоичной записи числа x единицы на нули в порядке убывания степени двойки, а значит после его завершения x станет равен нулю.

Поскольку ограничения задачи не позволяли для любого x делать 60 запросов, требовалось улучшить это значение. В частности, можно было бинпоиском найти максимальную степень двойки, не превосходящую x.

Для этого сделаем двоичный поиск по i в интервале $[0, 60)$. Если на запрос перевода 2^m мы получаем ответ `accepted`, переместим указатель на левую границу ($l = m$), иначе — на правую границу ($r = m$). Поскольку в процессе выполнения таких запросов x мог изменяться, не факт, что мы найдем именно максимальную степень двойки, не превышающую его, но можно доказать следующее **утверждение:** если после такого двоичного поиска выполнено $x \leqslant 2^t$, то так же выполнено $l \leqslant t$.

Чтобы доказать это утверждение, достаточно посмотреть на последний раз, когда мы получили `accepted`. Если после этого были получены только `rejected`, это значит, что ни для какого $m > l$, x не превышает 2^m. Это ровно то, что мы хотели доказать.

Осталось только повторить базовое решение, только начиная не с $i = 59$, а с $i = l$. Так как x не увеличился, l не превышает двоичного логарифма исходного x, а весь бинпоиск занял не более 6 операций, так как $60 < 2^6$. Следовательно, такой алгоритм переводит все деньги, даже имея в запасе 4 операции по сравнению с ограничениями в условии.

Задача E. Гениальная прогулка

Задачу можно решить используя модификацию алгоритма Дейкстры. Заметим, что для каждой вершины нас, как и в алгоритме Дейкстры, интересует лишь минимальный момент времени, в который в неё можно прийти. И при этом, для каждого ребра i из u_i в v_i можно вычислить минимальный момент времени, в который можно закончить переход по нему и оказаться в v_i, если минимальный момент времени, в который можно оказаться в u_i это x. Если $a_i < d_i$, по ребру пройти невозможно. Иначе, если можно начать переход в момент времени x, ответом является $x + d_i$. Иначе, нужно подождать первого момента, когда дождь закончится, и начать переход. Тогда ответом будет $\lceil \frac{x}{a_i+b_i} \rceil \cdot (a_i + b_i) + d_i$. Переход по ребру в момент x можно начать, если выполняется условие $x \bmod (a_i + b_i) + d_i \leqslant a_i$.

Задача F. Сэм и хранилище

Задачу можно решить, используя метод динамического программирования. Пусть $dp[i]$ — ответ на задачу, если игра происходит на массиве $a[i \dots n]$, а $dp[n+1] = 0$. Тогда $dp[i] = \max\limits_{j=i}^{n} a[j] - dp[j+1]$. Таким образом, мы получили решение за $O(n^2)$. Теперь заметим, что значения $(a[j] - dp[j+1])$, из которых мы выбираем максимум, не зависят от i, а значит не меняются при уменьшении i. Поэтому, можно сохранять эти значения в какой-нибудь структуре данных, и потом вычислять значение $dp[i]$, найдя максимум среди сохраненных значений на отрезке $[i, n]$. Заметим, что мы всегда находим максимум на префиксе, длина которого каждый раз увеличивается на 1. Поэтому вместо структуры данных, можно просто поддерживать максимум из значений.

Задача G. Постройка дороги

Если $s = 1$, то количество ходов, которое сделают игроки фиксировано и равно $n \cdot m$. Тогда Сэм выиграет только если это число нечётное.

Теперь считаем, что $s > 1$. Если одна из сторон нечётная или $s \geqslant 4$, то Сэм может первым ходом построить прямоугольник, симметричный относительно центра поля. После этого он может ходить центрально-симметрично ходам второго игрока. Значит, если второй игрок сделал ход, его сможет сделать и Сэм. Значит, Сэм выиграет.

Если же ни одно из этих условий не выполняется, то есть обе стороны чётны и $2 \leqslant s \leqslant 3$, то никакой игрок своим ходом не может построить прямоугольник, содержащий две центрально-симметричные клетки. Поэтому второй игрок может делать ходы центрально-симметричные ходам Сэма. Значит, Сэм проиграет.

Задача H. Робот-доставщик

Заметим, что:

1. После выполнения первой операции точка (x, y) перейдёт в точку $(y, -x)$.
2. После выполнения второй операции точка (x, y) перейдет в точку $(-y, x)$.
3. После выполнения третьей операции точка (x, y) перейдёт в точку $(1 + y, -(x - 1))$.
4. После выполнения четвертой операции точка (x, y) перейдёт в точку $(1 - y, x - 1)$.

После выполнения каждой из операций чётность суммы координат не меняется. Поэтому мы точно не сможем дойти из стартовой точки в конечную, если у них разная чётность суммы координат. Докажем, что в остальных случаях мы сможем построить требуемый путь. Для этого предъявим последовательности команд, которые из точки (x, y) переходят в точки $(x + 1, y + 1)$, $(x + 1, y - 1)$, $(x - 1, y + 1)$ и $(x - 1, y - 1)$. С помощью таких переходов можно построить путь в любую точку с такой же чётностью суммы координат.

1. После выполнения программы «23» точка (x, y) сначала перейдёт в точку $(-y, x)$, а потом в точку $(x + 1, y + 1)$.
2. После выполнения программы «14» точка (x, y) сначала перейдёт в точку $(y, -x)$, а потом в точку $(x + 1, y - 1)$.
3. После выполнения программы «32» точка (x, y) сначала перейдёт в точку $(1 + y, -(x - 1))$, а потом в точку $(x - 1, y + 1)$.
4. После выполнения программы «41» точка (x, y) сначала перейдёт в точку $(1-y, x-1)$, а потом в точку $(x - 1, y - 1)$.