Задача A. Выбросить середину

Давайте переберем позицию i такую, что элементы с 1 по i все останутся в массиве и будут первым куском, суммы которых равны (то есть массив, оставшийся после удаления подотрезка, будет делиться на два куска с одинаковой суммой как раз по позиции i). Попробуем теперь определить, можем ли мы в оставшейся части вырезать подотрезок, чтобы сумма остальных элементов была $sum = \sum\limits_{j \leqslant i} a_j$.

Нам нужно проверить в оставшейся части, существует ли подотрезок, сумма которого равна $x = S - 2sum$, где S это сумма всего массива. Это можно сделать с помощью двух указателей, в силу того, что массив содержит неотрицательные числа. Первый указатель будет индексом начала удаляемого отрезка, а второй будет двигаться только вправо, пока сумма удаляемого отрезка меньше x. Если мы нашли хотя бы один такой отрезок, то выводим его в качестве ответа. Получается решение со сложностью $O(n^2)$.

Задача B. Расставить ладьи

Задача может быть решена перебором. Для его ускорения, стоит проделать некоторые оптимизации. В начале выполняется предварительная обработка, в ходе которой для каждого поля, на котором находится кубик, определяется, сколько свободных полей или полей, на которых уже установлена ладья, находится в его соседстве. Если это число совпадает с числом на фишке, то ладьи устанавливаются и на остальных свободных полях. При установке ладей помечаются все свободные поля, до которых может дотянуться эта фигура (это позже позволит нам эффективно проверять, может ли какая-либо ладья достичь каждого свободного поля). После этого проверяется, существует ли поле, на котором находится кубик, такое что число соседних полей, которые свободны или на которых находится ладья, меньше числа на фишке. Если такое поле существует, то невозможно расставить ладей на этой доске так, чтобы соблюдались все ограничения.

Теперь будем расставлять оставшихся ладей. Будем пробовать расставлять ладьи в клетки по очереди, рекурсивно. Обработка следующего поля осуществляется вызовом функции, которая в качестве аргументов получает номер строки и столбца, в котором находится это поле, а в качестве результата возвращает информацию о том, успешно ли обработан/заполнен остаток доски. Если поле, которое мы в данный момент обрабатываем, занято или на него нельзя установить ладью, переходим к следующему полю. В противном случае для этого поля пробуются два варианта. Первый вариант – установить ладью на это поле, пометить поля, до которых она может дотянуться, и продолжить обработку следующего поля на доске. Эта обработка выполняется путем рекурсивного вызова функции обработки поля, которая в качестве аргументов получает координаты (номер строки и столбца) обрабатываемого поля. Если остаток доски успешно заполнен, второй вариант не рассматривается, и выполнение прерывается с возвратом информации о том, что доска успешно заполнена. В противном случае, если остаток доски не был успешно заполнен, рассматривается второй вариант, при котором ладья не устанавливается на это поле, и переходим к следующему полю доски с помощью рекурсивного вызова.

Чтобы дополнительно ускорить процесс, можно при каждом вызове функции для обработки поля проверять, существует ли кубик такой, что сумма числа установленных ладей на соседних полях и числа свободных (и не битых!) соседних полей меньше числа на кубике. Если такой кубик существует, невозможно расставить ладей согласно требованиям, поэтому перебор прерывается и возвращается назад.

Задача C. Максимальная префиксная сумма

Пусть $pref$ — массив префиксных сумм массива a, то есть $pref_i = a_1 + a_2 + \ldots + a_i$. Заметим, что тогда $f(a_l, a_{l+1}, \ldots, a_r) = \max(pref_{l-1}, pref_l, \ldots, pref_r) - pref_{l-1}$.

Тогда наша задача разбивается на две:

- Посчитать $\sum\limits_{l=1}^{n} \sum\limits_{r=l}^{n} \max(pref_{l-1}, pref_l, \ldots, pref_r)$, то есть посчитать сумму максимумов по всем подотрезкам массива $pref$, длина которых больше единицы. Для этого давайте для каждого элемента $pref_i$ найдем два значения:

1. lef_i — наибольший индекс слева ($lef_i < i$) такой, что $pref_{lef_i} > pref_i$ (или -1, если такого индекса нет).
2. $righ_i$ — наименьший индекс справа ($righ_i > i$) такой, что $pref_{righ_i} \geqslant pref_i$ (или $n+1$, если такого индекса нет).

Тогда заметим, что элемент $pref_i$ будет правым максимумом на всех подотрезках, где $lef_i < l \leqslant i$ и $i \leqslant r < righ_i$. Всего таких подотрезков будет $(righ_i - i) \cdot (i - lef_i)$. Также нужно будет вычесть единицу, так как мы исключаем подотрезки длины 1, и домножить на $pref_i$ при подсчете суммы.

- Посчитать $\sum\limits_{l=1}^{n}\sum\limits_{r=l}^{n} -pref_{l-1}$. Эта сумма равна $\sum\limits_{l=0}^{n} -(n-l)\cdot pref_l$.

Массивы lef и $righ$ можно найти с помощью линейных алгоритмов с помощью стека. Решение будет работать за $O(n)$.

Задача D. Сравнение квадратных корней

Необходимо было внимательно произвести сравнения, не приходя к вычислению вещественных чисел, учитывая знаки величин. Наивный математический способ использует величины четвертой степени от значений a, b, c, d. Можно было использовать int128 для поддержания абсолютных значений, либо использовать чуть больше условных операторов (понятно, что $\sqrt{b} \leqslant \sqrt{10^9}$, $\sqrt{bd} \leqslant 10^9$, $4bd \leqslant 4 \cdot 10^{18}$.

$$a + \sqrt{b} \quad vs \quad c + \sqrt{d}$$
$$a - c \quad vs \quad \sqrt{b} - \sqrt{d}$$

смена знака если выражения одного знака и отрицательные

$$(a-c)^2 \quad vs \quad d + b - 2\sqrt{bd}$$
$$(a-c)^2 - (b+d) \quad vs \quad -2\sqrt{bd}$$

проверяем знак слева и меняем знак

$$(a-c)^4 + (b+d)^2 - 2(b+d)(a-c)^2 \quad vs \quad 4bd$$

Задача E. Художники Баянистана

Первый шаг в нашем решении — построение heavy-light декомпозиции. Подвесим дерево в произвольной вершине, и для каждой вершины v вычислим размер её поддерева $s(v)$ (т.е. количество вершин, которые оно содержит). Теперь пусть v — произвольная вершина. Если у v есть ребенок с такой, что $s(c) \geqslant s(v)/2$, то ребро $v - -c$ называется тяжелым, в противном случае оно называется легким. Обратите внимание, что у каждой вершины может быть не более одного ребенка, соединенного тяжелым ребром. Таким образом, тяжелые ребра в дереве образуют набор непересекающихся путей.

Теперь мы можем взять всё дерево и разложить его на не более чем $n-1$ непересекающихся по ребрам путей следующим образом: пусть S — это множество вершин, у которых нет детей, соединенных тяжелым ребром. Для каждой из этих вершин мы строим один путь следующим образом: мы начинаем двигаться к корню, останавливаемся после того, как используем первое легкое ребро, и берем все ребра, которые мы использовали, чтобы сформировать один путь. Мы будем называть эти пути важными путями и нумеровать их от 1 до p. Что дает это разложение?

Представьте, что вы движетесь вниз по дереву, начиная с корня. Сколько раз вы можете использовать легкое ребро? Очевидно, что верхней границей будет $logn$, потому что каждый раз, когда вы используете легкое ребро, количество вершин в вашем поддереве как минимум уменьшается вдвое.

Мы только что показали, что для любой вершины v путь от v до корня содержит не более $logn$ легких ребер. Теперь представьте, что по мере того как мы идем по пути от v к корню, мы отслеживаем важный путь, по которому находимся. Он изменяется только тогда, когда мы используем легкое ребро. Поскольку на нашем пути не более $logn$ легких ребер, мы меняем важный путь не более $logn$ раз.

Почти то же самое верно и для пути между двумя произвольными вершинами v и w: пусть x — их ближайший общий предок в корневом дереве. Путь от v до w можно разделить на два пути: от v до x и от x до w. Каждый из этих двух путей использует ребра не более чем из $logn$ важных путей, следовательно, весь путь от v до w пересекает не более $2logn$ важных путей.

Для каждого важного пути мы будем хранить дерево отрезков. В каждой вершине дерева мы будем хранить для каждого цвета сколько ребер окрашены в этот цвет. Чтобы обработать запрос вида посчитать число перекрашиваемых ребер, мы рекурсивно спускаемся по дереву и суммируем сохраненную информацию. Чтобы обработать запрос перекрашивания, мы снова рекурсивно спускаемся по дереву и обновляем информацию, используя массовые операции.

Таким образом, чтобы ответить на запрос, сначала мы определяем отрезки важных путей, которые он использует. Это можно сделать за $O(logn)$. Затем мы обрабатываем каждый такой отрезок отдельно. Обработка каждого отрезка включает два запроса к дереву отрезков для его важного пути, следовательно, общая временная сложность составляет $O(log^2 n)$ на запрос.

Задача F. Блинчиковая Физтеха

У нас существует всего 3^n возможных башен. Для каждой башни существует n способов получить новую башню указанным в условии способом, а значит у нас есть граф из 3^n вершин и $3^n n$ ребер. Каждое ребро имеет единичный вес, и нам нужно для вершины из ввода находить кратчайший путь до некоторой хорошей вершины, которая соответствует хорошей башне.

Давайте сначала определимся, как пронумеровать наши башни. Пусть каждой башне соответствует многочлен вида $\sum s_i \cdot 3^i$, где i это номер блина сверху, а $s_i \in \{0, 1, 2\}$ задает вкус блина. Теперь отметим все хорошие башни, что для них расстояние 0. Хорошие башни можно получить, перебрав количество блинов вкуса A и вкуса B на вершине башни. Все эти вершины положим в очередь в качестве стартовых для BFS и запустим обход в ширину. Конечно, ребра хранить не нужно, нужно перебирать переходы тогда, когда обрабатываем вершину из очереди.

Теперь мы можем отвечать на каждый запрос за $O(1)$, просто посмотрев на значение расстояния для соответствующей вершины, определенное обходом в ширину. Решение получилось за $O(3^n n)$.

Задача G. Очередь за подарками

Ключевое наблюдение: мы всегда можем вернуть имя обратно в очередь таким образом, чтобы оно всегда использовалось для вычеркивания имени из списка, когда оно появится снова. Это подразумевает, что $m \leqslant operations \leqslant m + n$.

Тогда мы сначала можем промоделировать для каждого элемента в списке, какая проверка будет вычеркивать соответствующих элемент (будем отмечать в массиве свободных студентов, то есть тех, кто уже подходил). Таким образом мы поймем для каждого студента, каким по счету он должен подходить на проверку (это будет список его подходов). Пусть студент участвует в проверке i, а в следующий раз он должен участвовать в проверке $next[i]$. Чтобы определить, на какое место ему нужно вставать в очереди, нужно знать, сколько различных студентов будут подходить в моменты $(i, next[i])$. Это можно поддерживать в дереве отрезков или в ordered_set.

Задача H. Сгенерированная последовательность

Обозначим операцию сложения по модулю n как $\oplus$. Мы скажем, что шаблон w циклически встречается в тексте c на позиции i, если $w[k] = c[i \oplus k]$ для всех $k = 0, \ldots, m-1$. Циклическое вхождение, которое не является стандартным вхождением, мы назовем избыточным. В примере из условия задачи w встречается в c три раза, тогда как циклически встречается четыре раза. Циклическое вхождение, начинающееся на предпоследней позиции, является избыточным.

Посмотрим на множество хороших начал отрезка для k-го символа строки w. Введем $P(k)$ следующим образом:

$$P(k) = \{i : w[k] = c[i \oplus k]\}$$

.

В примере $P(0) = \{0, 2, 4, 5, 7\}$, $P(1) = \{0, 2, 5, 7\}$, $P(2) = \{0, 2, 3, 5, 7\}$. Таким образом, множество позиций циклических вхождений w равно $\bigcap_{k=0}^{m-1} P(k)$. Мы будем вычислять множество циклических вхождений и удалять из него избыточные.

Теперь мы займемся такой репрезентацией множеств $P(k)$, чтобы пересечения этих множеств (в соответствующем порядке) и подсчет избыточности были операциями, выполняемыми как можно быстрее. Определим $v_i = (ai + b) mod n$, тогда $c[i] = 0$ тогда и только тогда, когда $v_i < p$. Напомним, что a и n взаимно просты. Первый факт, который мы используем, что числа $v_0, v_1, \ldots, v_{n-1}$ попарно различны. Доказывается легко от противного (противоречие в нахождении общего делителя для взаимно простых a и n).

Получается, v является перестановкой чисел от 0 до $n - 1$. Однако это не совсем произвольная перестановка. Чтобы это заметить, давайте подумаем о том, как выглядит обратная перестановка к v, т.е. последовательность C_i, обозначающая позицию, на которой в последовательности v стоит значение $i (v[C_i] = i)$.

Утверждение. $C_k = ((k - b) \cdot a^{-1}) \ mod \ n$, а следовательно, последовательность C является циклической арифметической последовательностью по модулю n с разностью $a^{-1} \ mod \ n$.

Доказательство. C_k равно такой величине i, для которой $ai + b \equiv k \ (mod \ n)$. Искомое значение i действительно равно $((k - b) \cdot a{-}1) \ mod \ n$. Это означает, что $C_{k \oplus 1} - C_k = a^{-1} \ mod \ n$.

Рассмотрим пример из условия задачи. В этом случае последовательность v это последовательность

$$6, 2, 7, 3, 8, 4, 0, 5, 1$$

. Последовательность C является циклической арифметической последовательностью по модулю 9 с разностью $r = 5^{-1} \ mod \ 9 = 2$:

$$\underline{6 \to 8 \to 1 \to 3} \to \overline{5 \to 7 \to 0 \to 2 \to 4}$$

Заметим, что 4+2 = 6 (первый элемент цикла является следующим для последнего). На позициях в c, соответствующих первым четырем элементам последовательности C, находятся символы 0, а на следующих пяти символы 1.

Анализ последовательности C приводит нас к элегантной характеристике множеств $P(k)$. Утверждение. $P(k)$ является арифметической последовательностью по модулю n с разностью $a^{-1} \ mod \ n$.

Доказательство. Из определения множества $P(k)$ имеем: $P(k) = \{i : w[k] = c[i]\}$. Для $j \in \{0, 1\}$ обозначим через $S^j = \{i : c[i] = j\}$ множество позиций текста c, на которых находится цифра j. Тогда формула для $P(k)$ принимает вид $P(k) = S^{w[k]} \ominus k$.

Мы знаем, что S^0 является префиксом последовательности C, а S^1 является суффиксом последовательности C. Каждое из них является арифметической последовательностью по модулю n с разностью $a^{-1} \ mod \ n$, так что вычитание из всех его членов одного и того же числа k по-прежнему оставляет такую же последовательность.

Вернемся к примеру из условия.

$\mathbf{P}(0) = \{i : \mathbf{c}[i] = w[0] = 1\} = \{5, 7, 0, 2, 4\}$

$\mathbf{P}(1) = \{i : \mathbf{c}[i \oplus 1] = w[1] = 0\} = \{6, 8, 1, 3\} \ominus 1 = \{5, 7, 0, 2\}$

$\mathbf{P}(2) = \{i : \mathbf{c}[i \oplus 2] = w[2] = 1\} = \{5, 7, 0, 2, 4\} \ominus 2 = \{3, 5, 7, 0, 2\}$

Тогда ответом будет $\mathbf{P}(0) \cap \mathbf{P}(1) \cap \mathbf{P}(2) = \{0, 2, 5, 7\}$.

С этого момента вместо множеств $P(k) = \{i : w[k] = c[i \oplus k]\}$ мы будем использовать множества $V(k) = \{v_i : w[k] = c[i \oplus k]\}$.

Другими словами, множество $V(k)$ обозначает диапазон индексов в последовательности C, соответствующих элементам множества $P(k)$. Множества $V(k)$ являются циклическими отрезками. Для нашего примера мы имеем: $V(0) = [4..8], V(1) = [4..7], V(2) = [3..7]$. В конечном итоге $V(0) \cap V(1) \cap V(2) = [4..7]$.

Вместо пересечения множеств P(k) мы будем пересекать множества V(k). Определение пересечения множеств V(k) сводится к последовательному выполнению пересечений отдельных циклических отрезков, что легко, если результат также является циклическим отрезком. В нашем случае порядок вычисления пересечений имеет значение. Мы воспользуемся следующим очевидным наблюдением.

Пересечение двух циклических интервалов множества $[0..n-1]$, сумма длин которых не превышает n, является единственным циклическим интервалом.

В алгоритме мы начинаем с интервала $[0; n-1]$, а затем последовательно пересекаем его с интервалами $V(k)$, где $w[k] = a$ и символ a реже встречается в тексте c. Поскольку a является реже встречающимся символом, длина $V(k)$ равна не более чем $\frac{n}{2}$, то есть сумма длин таких интервалов не превышает n. Затем мы последовательно пересекаем результат с множествами $V(k)$ для позиций k, содержащих более частый символ.

Из вышеуказанного наблюдения следует, что результатом всегда будет циклический отрезок. Размер последнего отрезка дает нам количество циклических вхождений (мы помним, что каждой величине v соответствует ровно одна позиция, на которой в тексте встречается такая величина).

Остается еще проблема вычисления количества избыточных вхождений. Мы можем решить ее, проверяя для каждого $i > n-m$ (то есть для позиций, близких к концу текста c), принадлежит ли v_i полученному ранее результирующему отрезку. Количество позиций, для которых это выполняется, равно количеству избыточных вхождений. Таких проверок будет m, и они просты, поскольку мы проверяем, принадлежит ли какое-либо число данному циклическому интервалу.

Таким образом, мы получили решение всей задачи со сложностью $O(m+logn)$ ($O(logn)$ возникает из вычисления обратного к a по модулю n).

Задача I. Простые пятизначные числа

С помощью решета Эратосфена предподсчитаем все простые числа в требуемом диапазоне, их всего 8368. Тогда у нас существует $7 \cdot 10^7$ пар простых чисел из этого диапазона. Назовем суммой двух простых чисел кортеж (длиной 5) из суммы цифр в соответствующих разрядах. Давайте сохраним, какие суммы двух чисел мы можем получить из двух простых чисел (их всего $< 20^5$).

Теперь научимся быстро отвечать на запрос. Будем перебирать одно простое число и определять, существует ли у нас еще два простых числа, у которых кортеж (длиной 5) из суммы цифр в соответствующих разрядах равен запросу (за вычетом того самого перебираемого простого числа. Наши кортежи можно захешировать (каждый элемент кортежа <20), и хранить в массиве отображение из хеша в пару индексов простых чисел.

Задача J. Онлайн-кинотеатр

$dp[i]$ - минимальное количество отрезков чтобы покрыть префикс длины i попробуем пересчитать:

Считаем $dp[i+1]$ пусть $a[i] = 1$ (для 2 аналогично) тогда мы обязаны покрыть i-ый элемент, но тогда мы обязаны взять отрезок того типа, которому принадлежит $a[i]$ и при этом очевидно его стоит закончить в i (сдвинем влево) но тогда мы покроем все $1/B$ на отрезке от $i-k+1$ до i если на отрезке $i-k+1$ до i нет 2, то мы свободно пересчитываемся через $dp[i-k]$ но что если на отрезке от $i-k+1$ до k есть 2 тогда если её позиция это j и $dp[j+1] = x$ попробуем пересчитать $dp[i+1]$ как $x+1$ (ответ для покрытия j и всего слева + один отрезок типа 1 который мы добавили) правильный ли наш пересчёт? очевидно нет, тк $dp[j+1]$ никак не учитывает что все 1-ки покрыты начиная с $i-k+1$ НО!!! легко понять что $x \leqslant dp[x] \leqslant x+1$ тк если $dp[x] \leqslant x$, то этот же ответ покрывает и префикс с j, значит и $dp[j+1] \leqslant x$ но также у нас есть пример и на $x+1$, значит нам осталось лишь понять что выбрать: $dp[i] = x$ или $dp[i] = x+1$

Для решения этой проблемы мы модифицируем дп до $dp[i].ans/have1/have2$ где $dp[i].ans$ это ответ на префиксе i, а $have1$ и $have2$ максимально насколько может торчать отрезок типа $1/2$ так, чтобы оптимальный ответ не изменился. Пример: $b11111$ при $k = 3$ и префиксе $i = 5$ понятно что $dp[i].ans = 2$ но если даже последние 2 единички уже покрыты, ответ так и останется 2. Таким образом $dp[i].have1 = 2$ (но если торчит 3, то ответ уже изменится на 1)

Теперь вполне понятно как пересчитать $dp[i+1].ans$ при $a[i] = 1$ нужно взять отрезок 1, взять самую правую 2, и проверить, правда ли то что отрезок "покрытых 1, торчащих за 2"не изменит ответ если не изменит, то $dp[i+1].ans = x+1$ иначе $dp[i+1].ans = x$

Но теперь появилась новая проблема: нам нужно пересчитать $have1$ и $have2$. Заметим что это монотонная функция, т.е. если отрезок торчит на k и ответ не изменился, то и для всех меньших ответ не менялся, значит сделаем по $have1$ и $have2$ бинпоиски и зная насколько из префикса $i+1$ торчит отрезок определённого типа мы сможем проверить, меняется ли ответ

И последний случай если $a[i] = B$: просто пересчитаемся 2 раза, как если бы мы вставили отрезок типа 1 или типа 2

Итого $O(nlogn)$ времени и $O(n)$ памяти

Задача K. Черно-белый граф

Сначала мы создаем дополнительную вершину X и соединяем её со всеми с нечетной степенью. Это не оставляет вершин с нечетной степенью. Это означает, что граф является эйлеровым, или, если он не связен, что каждая из его компонент является эйлеровой. Теперь мы находим эйлеровы циклы во всех компонентах и распределяем права на строительство между ними, чередуя 1 и 2.

Для каждой компоненты с хотя бы одной вершиной нечетной степени это жизнеспособное решение. Для компонент без нечетных степеней есть два случая:

- Одна степень больше или равна 4. Начиная с такой вершины, формируется корректное решение.
- Все степени равны 2. Компонента является циклом. Если она содержит четное число вершин, это возможно. В противном случае решить этот граф невозможно.

Задача L. Посадка в самолет

Будем последовательно для каждого пассажира, начиная с пассажира n, вычислять время, когда он сядет на свой ряд. Давайте сохранять точки, в которых пассажир будет вставать в пробку, то есть будем хранить набор пар (a, b), которые обозначают, что координату a пассажир может пройти не раньше момента времени b.

Для пассажира n набор содержит только $(0, 0)$. Для пассажира $n - i$ набор будет содержать как минимум $(-i, 0)$. Если пассажир i ограничен парой (a, b), то он достигнет места $r[i]$ в момент времени $b - a + r[i]$ и сядет в момент времени $b - a + r[i] + t[i]$. Мы можем найти первое время, когда пассажир i может достичь места $r[i]$, поиском в наборе пары (a, b), такой что $a < r[i]$ и $b - a$ максимально.

Чтобы поддерживать набор при переходе от одного пассажира к другому, мы просто смотрим на все пары, мимо которых прошел последний пассажир, и вычитаем 1 из позиции. Другими словами: если пассажир i находится на месте 5 в момент времени 10, то пассажир $i - 1$ не может находиться за местом 4 в момент времени 10.

Это сразу дает решение за $O(n^2)$. Мы будем хранить набор в виде массива пар. Для пассажира i ищем пару (a, b) с максимальным значением $b - a$, удовлетворяющую условию $a < r[i]$. Вычисляем время, когда пассажир сядет ($v[i] = b - a + r[i] + t[i]$). Затем вставляем пару $(r[i], v[i])$ в список (пассажир i должен находиться на месте $r[i]$ в момент времени $v[i]$). Чтобы перейти к пассажиру $i - 1$, мы выполняем обновление диапазона, заменяя все (a, b), такие что $a \leqslant r[i]$, на $(a - 1, b)$. Затем мы выводим максимальное значение v среди всех пассажиров.

Решение за $O(n \log n)$ хранит набор в виде монотонной очереди. Заметим, что нам не нужно хранить две пары (a, b) и (c, d), если $a < c$ и $b - a > d - c$: пара (c, d) никогда не будет влиять на ответ для последующих пассажиров. Более того, в любом подмножестве пара (a, b) с максимальным значением $b - a$ это пара с максимальным a.

Чтобы поддерживать монотонную очередь, после вставки пары (a, b) удаляем все пары (c, d) с $c \geqslant a$ и $d - c \leqslant b - a$. Нам понадобится структура данных, которая сможет поддерживать поиск, вставку и обновление диапазона. Сбалансированное двоичное дерево поиска (например, декартово дерево) может обрабатывать каждую из этих операций за $O(logn)$ времени, что дает общее время выполнения $O(nlogn)$.

Задача M. Life on Mars?

Заметим, что сумму квадратов арифметической прогрессии можно посчитать за $O(1)$.

$(Cn + A) \bmod M$ имеет период $\frac{M}{\gcd(C,M)}$, причём значение обходит все остатки, сравнимые с A по модулю $\gcd(C, M)$. Таким образом, C можно взять по модулю $\frac{M}{\gcd(C,M)}$, при этом каждый период вносит в ответ некоторую сумму квадратов арифметической прогрессии. Теперь $C < M$.

Вычтем из A остаток по модулю $\gcd(C, M)$, прибавив $K \cdot (A \bmod \gcd(C, M))$ к ответу. Теперь C, A и M делятся на $\gcd(C, M)$, поделив, получим в $\gcd(C, M)^2$ раз меньший ответ. Теперь $\gcd(C, M) = 1$.

Если бы C было небольшим, то хорошо бы работало решение за $O(\frac{K \cdot C}{M})$. Действительно, если для некоторого отрезка n $(Cn+A) \bmod M$ не переполняется, то вклад этого отрезка в ответ — сумма квадратов арифметической прогрессии.

Пусть мы нашли такое r, что r невелико и $H := Cr \bmod M$ невелико. Тогда мы можем аналогично посчитать сумму для всех n вида kr за $O(\frac{K/r \cdot H}{M})$.

Таким же образом можно посчитать ответ для n со всеми остальными остатками $\bmod r$. Отсюда получаем решение, работающее за $O(r + \frac{K \cdot H}{M})$.

Почему найдётся подходящее r? Рассмотрим $C, 2C, 3C, \ldots, \lceil\sqrt{M}\rceil C$ по модулю M. Какие-то два из них по принципу Дирихле будут отличаться не больше чем на $\sqrt{M}$: $(iC - jC) \bmod M = O(\sqrt{M})$. Если $i > j$, возьмём $i - j$ в качестве r, иначе заменим в задаче A на $(A + (k-1)C) \bmod M$ и C на $M - C$ и возьмём $j - i$ в качестве r.

Итоговая асимптотика решения — $O(\sqrt{M})$.

Отметим, что существует решение за $O(\log(M))$, использующее продвинутую версию алгоритма floor-sum. Для этого заметим, что $(Cn + A) \bmod M = Cn + A - M \cdot \lfloor\frac{Cn+A}{M}\rfloor$, после чего раскроем скобки и воспользуемся рекурсивными формулами для $f(a, b, c, n) = \sum_{x=0}^{n} \lfloor\frac{ax+b}{c}\rfloor$, $g(a, b, c, n) = \sum_{x=0}^{n} x\lfloor\frac{ax+b}{c}\rfloor$ и $h(a, b, c, n) = \sum_{x=0}^{n} \lfloor\frac{ax+b}{c}\rfloor^2$.