Задача A. Анизотропные простые числа

Так как делителей два (1 и p), то $p+1$ должно не делиться на 2, а это возможно только при чётном p. То есть единственным анизотропным простым числом является 2.

Задача B. Буду ли я участвовать?

Считаем, что по умолчанию ответ равен `+`.

Далее, если студент участвовал в 2+ финалах или 6+ региональных турнирах, ответ меняем на `-`. Иначе, если студент участвовал в 5 региональных турнирах, ответ меняем на `-` при $s > 8$ и на `?` в противном случае (в C/C++ это можно сделать тернарным оператором). Иначе, если год рождения студента меньше 2002, а год первого поступления меньше 2021, аналогичным образом меняем ответ на `-` или `?` в зависимости от значения s. После чего выводим ответ (во всех остальных случаях студент может участвовать).

Задача C. Везёт слабейшим

Приводим все числа к целым (умножая на 10). Сумма тем самым равна 1000, и теперь очевидно, как обойтись 1000 шарами. Затем ищем наибольший общий делитель («стоимость» одного шара) и делим все числа на него.

Задачу можно решить и без знания алгоритма Евклида, просто заметив, что общий делитель — это произведение степеней двойки и пятёрки. Поэтому проверяем, верно ли утверждение «все числа делятся на 2», пока оно верно, делим все числа из текущего набора на 2. Затем аналогичным образом действуем для 5. Сумма чисел после того, как процесс завершится, и будет ответом.

Задача D. Грузим файл

При p процентах закачки длина дуги равна $3.6 \cdot p$ градусов. За время закачки $(d \cdot p/100)$ точка s прошла расстояние $v \cdot d \cdot p/100$ градусов. Соответственно, прибавляем длину дуги к пройденному расстоянию и приводим координаты к диапазону $[0, 360)$ (например, в C/C++ вызов функции $fmod(x, 360.0)$ приводит неотрицательный x к нужному диапазону).

Задача E. Добиться делимости

Заметим, что остатки от деления на 7 степеней числа 10 периодичны с периодом 6 (1,3,2,6,4,5). Поэтому находим остаток от деления на 7 заданного числа. Если он равен 0, то изменение цифры на +-1 приведёт к тому, что на 7 число делиться не будет, и ответ 0. Иначе находим те номера разрядов (по модулю 6), в которых прибавление или отнимание единицы даст 0 по модулю 7 в результате изменений, после чего идём с шагом 6 и прибавляем по 1 за каждый такой разряд, не равный 0 (или единице в самой левой позиции) для вычитания и по 1 за каждый такой разряд, не равный 9, для сложения.

Задача F. Если заказчик требует...

Рассмотрим некоторый ответ — квадрат i_1, i_2, j_1, j_2. Назовем его S_1. Пусть, например, $i_2 - i_1 \neq 1$. Тогда рассмотрим квадраты S_2 и S_3 с координатами вершин i_1, i_1+1, j_1, j_2 и i_1+1, i_2, j_1, j_2 соответственно. Пусть для каждого из них сумма в "красных" квадратах не превосходит суммы в "синих" квадратах. Обозначим разности этих сумм за $\Delta(S_i)$. Нетрудно проверить, что $\Delta(S_1) = \Delta(S_2) + \Delta(S_3)$. Но тогда $\Delta(S_1) \leqslant 0$. Противоречие. Отсюда следует, что если в ответе одна из сторон больше 2, то ее можно уменьшить. Таким образом, достаточно проверить только квадратики 2×2.

Альтернативный вариант: заметить, что если посмотреть на входной массив как на массив частичных сумм, то нас спрашивают, существует ли прямоугольник с положительной суммой. Он существует тогда и только тогда, когда хотя бы один из элементов исходного массива положителен. Таким образом, достаточно построить массив частичных разностей и найти в нем положительный элемент (единственная тонкость — надо выкинуть нулевые строку и столбец).

Задача G. Ё-тауэр

Заметим, что, так как у нас нет нулей, то распределение цифр как * ** *** ... при движении вверх или ... *** ** * при движении вниз всегда даст монотонную последовательность. То есть номер последнего этажа ограничен сверху $O(\sqrt{n})$ знаков (длина последнего блока не больше минимального x, такого, что $x(x-1) \geqslant n$).

Будем решать задачу динамическим программированием. Варианты с движением вниз и вверх незначительно отличаются только в реализации, так что опишем общий подход.

Массив dp содержит пары чисел — начальную цифру записи номера минимального этажа и длину этой записи для префикса длины i. При переходе к $i+1$-й цифре мы проверяем, будет ли число, полученное при дописывании этой цифры к последнему этажу для уже вычисленных значений. При этом в случаях, когда длины нового числа и его «предшественника» не равны, при сравнении хватает сравнения только вторых элементов пары. К цифровой строке мы переходим только при равенстве. При этом, в соответствии с замечанием выше, длина строки не превосходит $\sqrt{i}$, то есть получаем в итоге $O(n\sqrt{n})$.

После вычисления минимумов для движения вниз и вверх сравниваем их и выдаём минимальный.

Задача H. Желаете подешевле?

Построим ориентированный граф, вершинами которого являются числа от 1 до 3999, а ребро из вершины a в вершину b существует тогда и только тогда, когда из числа a можно за одно действие (конкатенация или убирание префикса/суффикса) получить число b, и имеет длину, равную стоимости этого действия. Наиболее простым способом построения графа будет перебор всех пар римских чисел и проверка, получается ли из приписывания первого числа ко второму существующее римское число (это можно быстро сделать, построив два `map` — число по римской записи и римская запись по числу). Если получается, добавляем четыре ребра (два — с приписыванием справа или слева, и два — с удалением справа или слева). Также добавляем для стартовой вершины и каждого римского числа два ребра — от римского числа к пустой вершине удалением и от пустой вершины к римскому числу приписыванием.

После чего находим кратчайший путь в графе от пустой вершины до всех чисел с помощью алгоритма Дейкстры и отвечаем на запросы по предпросчитанному массиву кратчайших путей.

Задача I. Загадано основание

Задача допускает несколько решений.

Мы знаем, что 1 записывается одной цифрой. Попробуем найти число, которое записывается двумя цифрами, рассматривая числа, которые записываются не более чем двумя цифрами в любой подходящей системе счисления. Первое из таких чисел 3 (2^2-1), если оно однозначное, то теперь максимальное известное число, которое гарантированно записывается не более чем двумя цифрами – $4^2-1=15$, если однозначно и оно, то запросим $16^2-1=2^8-1=255$. Если в какой-то из этих трех запросов получаем ответ 0 (число двузначное), то получаем две границы, между которыми бинарным поиском ищем минимальное число, записываемое двумя цифрами – это и есть ответ. Иначе 16^2-1 записывается одной цифрой. Это будет левая граница бинпоиска. Ставим правую границу в 2^16, которое заведомо не однозначно, и ищем бинпоиском минимальное число, которое записывается двумя цифрами. Итого - 3 запроса изначально плюс заведомо не больше 16 на бинпоиск.

Второе решение — запрашивать $2^{(2^k)}$, начиная с $k=4$ до тех пор, пока не будет чётного количества знаков, затем запустить бинпоиск за k запросов. Получаем максимум 17 запросов (в случае, когда для 2^{16} количество знаков чётно). В этом случае требуется различить возможные варианты неоднозначностей b и b^{2n+1} (когда найденное основание является нечётной степени). Максимальное количество вариантов — 4 для 15-й степени, то есть двух запросов на разделение всех случаев хватает (например, для 15-й степени запрашиваем сначала b^4, затем или b, или b^5). Это решение, в отличие от первого, работало бы и до 2^{32} за 32+3 запроса (первому уже требуется 4 запроса).

Также существуют решения с бинпоиском на модифицированном множестве элементов, которые укладываются в меньшее количество запросов.

Задача J. Инфляция и трамвай

Ключевая идея задачи состоит в том, что оптимальный ответ — минимальную цену минимального остовного дерева – следует искать либо при курсе l, либо при r.

Покажем, почему это так. Поставим в соответствие каждому ребру прямую на координатной плоскости — ребру i соответствует прямая $b_i \cdot x + a_i$. Выберем некоторую точку $x \in [l, r]$ и рассмотрим набор $n-1$ прямых, которые образуют как рёбра минимальное остовное дерево для точки x. Предположим, у нас есть только такие прямые на плоскости. Тогда стоимость минимального остовного дерева в точке y будет вычисляться как сумма $b_i \cdot y + a_i$ по всем этим прямым. Однако сумма линейных функций линейна, то есть цена минимального остовного дерева представляет собой прямую, и её минимум находится в одном из краёв – в l или в r.

Однако, если мы сравним точки x и $x-1$ теперь(аналогично для x и $x+1$), то получим, что минимальное остовное дерево на оптимальном наборе прямых из точки $x-1$ для точки $x-1$ будет не хуже, чем на оптимальном наборе прямых из точки x для точки $x-1$. Двигаясь таким образом к l, мы покажем, что ответ в l для того набора прямых, который оптимален в l, будет не хуже ответов для оптимальных наборов в других точках. Двигаясь в другую сторону(к r), можно доказать аналогичное утверждение для r.

Чтобы найти минимальное остовное дерево для конкретной точки, можно использовать алгоритм Краскала. Он заключается в том, чтобы отсортировать все имеющиеся рёбра в порядке неубывания веса, и, проходясь по ним в таким порядке, добавлять текущее ребро в ответ, если оно не образует цикл. Чтобы отслеживать последнее условие, можно использовать структуру данных систему непересекающихся множеств.

Задача K. Йошкар-Ола — Москва

Сначала рассмотрим следующие замечания:

1. В операциях типа 2 вычтем 1 из k_i и будем использовать индексацию с 0.
2. Обозначим через $(v_1, v_2, \ldots, v_r)$ цены банок, которые встречались хотя бы раз в командах типа 1, отсортированные по возрастанию. Выполняется за $r \leqslant q$.
3. Задачу для операции типа 2 можно переформулировать как поиск минимального m, удовлетворяющего условию A: количество банок в вагоне m_i с ценностью $\leqslant v_k$ строго больше k_i.
4. К этой задаче можно применить двоичный поиск по r, решая задачу проверки условия за $O(\log r)$ операций.
5. Процесс двоичного поиска можно представить себе как перемещение по узлам, аналогичным структуре дерева Фенвика.

Будем использовать двумерное дерево Фенвика (возможна также реализация через дерево отрезков).

- Для каждого узла храним количество банок в вагоне j с ценностями из $[v_l, v_r]$.
- Операции типа 1: обновляем $O(\log r)$ узлов, добавляя w_i к соответствующим p_j.
- Операции типа 2: вычисляем p_{x_i} и перемещаемся по узлам ($O(\log r)$ раз

Кроме того, применим следующую оптимизацию:

- Используем дерево Фенвика для вычисления p_j.
- Операция типа 1: добавляем w_i к элементу l_i и вычитаем из элемента $r_i + 1$.
- Операция типа 2: $S =$ сумма элементов с 1 по x_i.
- Применяем сжатие координат для экономии памяти

Оценка сложности

- Команда типа 1: $O(\log^2 q)$ (обновление $O(\log q)$ узлов за $O(\log q)$ каждый)

- Команда типа 2: $O(\log^2 q)$ ($O(\log q)$ запросов суммы за $O(\log q)$ каждый)
- Общая сложность: $O(Q \log^2 Q)$

Задача L. Кузнечик и отмеченные точки

Рассмотрим величину $d_{i,j}$ — количество способов оказаться в точке x_i, пройдя не менее j препятствий.

Допустим, мы находимся в точке x_i и уже прошли как минимум j препятствий. При этом, если было пройдено $j_1 > j$ препятствий, то такие пути будут учтены $\binom{j_1}{j}$ раз. Это позволяет вычислить $d_{i,j}$ по следующему рекуррентному соотношению:

$$d_{i,k} = F[x_i] + \sum_{j<i} dp_{k,k-1} \cdot F[x_i - x_j],$$

где $F[x_i - x_j]$ — это число маршрутов из x_j в x_i. Это равно числу Фибоначчи, что можно посчитать с помощью возведения матрицы в степень $A^{x_i - x_t}$.

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Для оптимизации хранения и пересчёта значений, храним для каждого k сумму вида:

$$\sum d_{i,k} \cdot A^{-x_i},$$

то есть предварительно умножаем на обратную степень матрицы A.

Затем при пересчёте d для новой точки x_j мы умножаем сумму для фиксированного k на A^{x_j}.

Теперь посчитаем $a_k = \sum d_{i,k} \cdot F[l - x_i]$. Это аналогичная предыдущему определению величина, только рассматривающая все маршруты из 0 в l.

$$\sum f(\text{маршрут})^k,$$

отметим, что мы уже посчитали для каждого $i \leqslant k$ суммы следующего вида:

$$\sum \binom{f(\text{маршрут})}{i}.$$

Поскольку биномиальный коэффициент $\binom{f}{i}$ является многочленом от f степени i, зная значения сумм для всех $i \leqslant k$, можно восстановить суммы $\sum f^j$ для всех $j \leqslant k$.

Таким образом, последовательно восстанавливая значения в порядке от 1 до k, можно получить финальный ответ.

Задача M. Лист клетчатой бумаги

Сначала рассмотрим, как определить, можно ли полностью разрезать на домино последовательность a.

Преобразуем задачу в возможность выполнения следующих двух операций:

- Уменьшить любое a_i на 2.
- Уменьшить оба a_i и a_{i+1} на 1, при условии что $a_i = a_{i+1}$.

Очевидно, что это эквивалентно исходной задаче.

Заметим, что эти операции явно связаны с чётностью. Наша конечная цель — достичь $a_i = 0$. Используя Операцию 1, нам достаточно достичь $a_i \equiv 0 \pmod 2$. Точнее, нам нужно достичь $a_1 \equiv a_2 \equiv \ldots \equiv a_n \pmod 2$. Если временно игнорировать конечное ограничение $a_i \geqslant 0$, цель становится в максимизации получающегося $\min\{a_i\}$; если этот минимум $\geqslant 0$, то последовательность хорошая.

Это приводит к предварительной идее: для соседних $a_i \equiv a_{i+1} \pmod 2$ использовать Операцию 1, чтобы уменьшить их до $\min\{a_i, a_{i+1}\}$, после чего можно использовать Операцию 2. Более того, для $a_i = a_{i+1}$ мы можем почти считать, что эти два числа «исчезают», потому что когда мы хотим использовать Операцию 2, чтобы уменьшить a_{i+1} и a_{i+2} на 1, мы можем вместо этого применить операции к парам (a_{i-1}, a_i) и (a_{i+1}, a_{i+2}), при условии что $a_{i-1}, a_{i+2} \leqslant a_i = a_{i+1}$.

Это наводит на мысль о процессе, аналогичном сопоставлению скобок: последовательно устранять соседние пары с одинаковой чётностью, пока не останется один или ноль элементов. Мы можем использовать следующий жадный подход: пройти по последовательности, поддерживая стек, представляющий позиции элементов, которые ещё не были устранены. Это даёт набор сопоставлений. Легко доказать, что если в стеке останется хотя бы 2 элемента, то последовательность нельзя полностью удалить (разница между количеством элементов с $a_i + i \equiv 0 \pmod 2$ и количеством с $a_i + i \equiv 1 \pmod 2$ будет не менее 2, что является инвариантом, но в конце это значение должно быть 0 или 1).

Мы утверждаем: если решение существует, то существует решение, в котором после нескольких применений Операции 1 и некоторых Операций 2 числа между сопоставленной парой становятся равными. Конкретно, мы дополнительно храним в стеке максимальное значение, до которого были уменьшены промежутки между соседними элементами стека. Для вновь сопоставленной пары $a_i \equiv a_j \pmod 2$, если промежутки между ними были уменьшены до x, то необходимо $a_i \not\equiv x \pmod 2$. Затем все эти числа уменьшаются до $\min\{a_i, a_j, x-1\}$. Если в какой-то момент это число становится < 0, решения нет.

Нетрудно заметить, что для элемента $a_i > a_{i-1}, a_{i+1}$ нам всегда нужно использовать Операцию 1. После выполнения всех таких Операций 1 появится несколько сопоставленных пар $a_i = a_{i+1} > a_{i-1}$. Возьмём наименьшую такую 1 и применим Операцию 2, повторяя этот процесс. Легко доказать, что этот жадный процесс эквивалентен упомянутому выше. Используя простой аргумент замены, можно показать, что использование Операции 2 вместо Операции 1 в определённых случаях не влияет на ответ (ситуации, где Операция 1 применима, а Операция 2 нет, образуют нисходящую лестницу, но в основании всегда будут две Операции 2, что позволяет произвести простую корректировку).

Заметим, что это сопоставление образует древовидную структуру. Разворачивая описанный выше процесс проверки, конечное условие корректности вносит требование $a_i - \text{глубина}_i \geqslant 0$, т.е. при сопоставлении в позиции i требуется $a_i - \text{размер\_стека} \geqslant 0$. Это приводит к методу проверки за $O(n)$.

Более того, поскольку форма стека относительно фиксирована (только шаблоны вида «оканчивается на нечётное/чётное, длина k»), мы можем поддерживать $f(k, 0/1)$, представляющее количество стеков размера k с верхним элементом нечётным/чётным (количество начальных позиций). Вклад и ограничения каждого a_i на стек выглядят следующим образом:

- Для элемента стека, который не может быть сопоставлен, требуется $a_i - \text{размер\_стека} \geqslant 0$, при этом размер стека увеличивается на 1, а чётность верхнего элемента меняется.

- Для элемента стека, который может быть сопоставлен, требуется $a_i - \text{размер\_стека} + 1 \geqslant 0$, при этом размер стека уменьшается на 1, а чётность верхнего элемента меняется.

Таким образом, мы можем напрямую поддерживать два массива, представляющих соответствующие ответы, с поддержкой присваивания на отрезке (для k), глобального смещения, точечного изменения и точечного запроса. Операцию присваивания можно выполнять напрямую из-за непрерывности ответа (если существует стек размера k, то должен существовать стек размера $k - 1$). Общая сложность составляет $O(n)$.