Задача A. Луч смерти

Автор задачи: Ильдар Гайнуллин, разработчик: Антон Вдовин

У этой задачи много решений — `DCP Offline`, корневые декомпозиции и другие, но мы рассмотрим авторское решение, без сложных идей и структур.

Будем хранить все точки в `std::set`. Давайте поддерживать прямые между парами соседних точек в текущем множестве в другом `std::multiset`. При добавлении точки добавится и одна новая прямая (если это не единственная точка в множестве). При удалении точки может удалится две прямые и добавиться новая или же просто удалиться одна прямая. После каждого изменения множества точек достаточно посмотреть, что максимальный элемент в мультимножестве прямых совпадает с минимальным элементом (это означает, что все линии, которые мы провели между парами соседних по добавлению точек, являются одинаковыми, то есть все точки лежат на одной прямой). Для этого также нужно уметь хэшировать прямые (потому что разные тройки коэффициентов могут давать одни и те же прямые), что является общеизвестной задачей и остается на упражнение читателю.

Задача B. Пирамида Видений

Автор задачи: Даниил Орешников, разработчик: Степан Мишанин

Для решения задачи достаточно проэмулировать процесс, описанный в условии.

Создадим двумерный массив a размера $n \times n$, в который считаем нашу пирамиду (пусть на каждом уровне она будет дополнена справа нулями до размера n, мы на эти элементы при решении никогда смотреть не будем). Заметим, что кристалл, находящийся на позиции $a_{i,j}$, стоит на двух кристаллах, находящихся на позициях $a_{i+1,j}$ и $a_{i+1,j+1}$. Тогда пройдемся по всем кристаллам (за исключением, конечно же, кристаллов с последнего уровня): пусть сейчас мы рассматриваем кристалл $a_{i,j}$, из кристаллов на позициях $a_{i+1,j}$ и $a_{i+1,j+1}$ выберем кристалл с большим значением (при их равенстве — $a_{i+1,j+1}$) и уменьшим его значение на значение рассматриваемого кристалла. Тогда ответом является последняя строка нашего двумерного массива.

Задача C. Побег от зомби

Автор задачи и разработчик: Павел Скобелин

Для каких-то продвижений по задаче нужно было понять более простой способ вычисления функции f.

Покажем, что существует путь минимальной длины до точки (x, y), такой, что он состоит только из ходов длины 1.

Несложно показать, что любой ход на x, где $x > 2$, строго выгоднее заменить на 5 ходов: $1 + 1 + (x - 2) + 1 + 1$, так как

$$1^2 + 1^2 + (x-2)^2 + 1^2 + 1^2 = x^2 - 4x + 4 + 4 = x^2 - 4x + 8 < x^2$$

Далее нужно показать, что можно заменить все отрезки длины 2, и получить ответ не хуже. Но, на самом деле, это не было необходимо: Даже имея этот факт, несложно показать, что функция f выглядит так: $f(x, y) = 2 \cdot \max(|x|, |y|) - ((x + y) \mod 2)$. Это можно показать по индукции.

Далее, понимая, как выглядит функция, есть несколько решений. Первое решение — бинарный поиск. Для начала, нужно добиться такого, чтобы у загаданной точки координата по оси OX совпадала с координатой по оси OY. Как такое сделать? Для удобства, первым действием можно было передвинуть точку в первую четверть. После этого заметим, что если мы уменьшим меньшую координату на четное число, то наша функция никак не изменится, а если большую — то изменится. Значит, несложно за 1 запрос понять, какая координата больше. Далее, можно было делать бинарный поиск, чтобы понять, в какой момент функция начнет изменяться относительно изначальной позиции. То есть, если $f(x, y) = f(x + 2, y) = \cdots = f(x + 2k, y) < f(x + 2k + 2, y) < f(x + 2k + 4, y)$, значит, что $y \in [x + 2k, x + 2k + 2)$. После этого аккуратным разбором случаев можно было добиться уравнивания координат.

Теперь мы знаем, что $x = y$. Тогда, если мы будем поддерживать инвариант, будет выполняться $f(x, y) = 2 \cdot x$. Значит, нам нужно уменьшать x (и y), пока функция расстояния не начнет снова расти. Этот момент нужно найти бинарным поиском — несложно показать, что это единственный момент, в который координата точки будет $(0, 0)$.

Таким образом, решение работает за 2 бинарных поиска, требующих порядка 32 запросов, что легко укладывается в ограничения.

Задача D. Великая теорема Фестера

Автор задачи и разработчик: Даниил Орешников

Решим задачу при $n = 0$. В таком случае, для любого числа x, $x^n = x^0 = 1$. Тогда, ни для каких a, b, c не выполнится равенство:

$$a^0 + b^0 = c^0$$

$$1 + 1 = 1$$

Значит, в случае $n = 0$ ответа на задачу не существует.

А как найти ответ при $n = 1$?

$$a + b = c,$$

где c нам дано, и нужно найти подходящие a, b. Положим $a = c$, $b = 0$, и выполним условие задачи.

Задача E. Лабиринт Кошмаров

Автор задачи: Даниил Орешников, разработчик: Степан Мишанин

Если говорить более формально, то в задаче от нас требуется найти гамильтонов цикл в дереве, в которое добавили ребра между всеми вершинами на расстоянии не больше трёх. Очевидно, что явно построить такой граф, дополнив дерево ребрами мы не можем, ведь в худшем случае такой граф будет полным, — но для решения это и не потребуется, просто в дальнейшем будет считать, что из текущей вершины нашего дерева можно переместиться в любую вершину на расстоянии не больше трёх.

Покажем теперь, как можно в нашем дереве найти гамильтонов путь, начинающийся в корне и заканчивающийся в одном из его детей. Решать эту задачу будем рекурсивно. Пусть сейчас мы рассматриваем вершину v, у которой имеется k детей $u_1, u_2, u_3, ...u_k$. Будем по очереди рассматривать всех её детей, тогда у нас возможны два случая:

- Если у рассматриваемой вершины u_i есть дети, будем по очереди перемещаться в них (мы это можем сделать, так как расстояние между v и любым ребёнком u_i равно двум) и решать задачу рекурсивно для них. Напомним, что мы ищем гамильтонов цикл, начинающийся в корне и заканчивающийся в одном из его детей. Тогда, решив задачу для какого-то ребёнка t_m вершины u_i, мы окажемся в каком-то из детей t_m и можем из него переместиться в следующего ребёнка u_i, так как расстояние между ними будет равно трём. В конце остаётся только переместиться в вершину u_i, что также возможно, так как расстояние между u_i и любым ребёнком её ребёнка равно двум.

- Если у рассматриваемой вершины u_i нет детей, то добавим u_i в ответ будто бы мы зашли в неё и продолжим рассматривать детей v. Тогда если у следующего ребёнка u_j тоже нет детей, мы можем переместиться в него, так как расстояние между u_i и u_j равно двум, иначе — следуя описанному в первом пункте, мы переместимся в какого-то ребёнка u_j, что также возможно, так как расстояние между u_i и ребёнком u_j равно трём.

С помощью описанного способа найдем гамильтонов путь в нашем дереве. Так как он завершится в каком-то из детей корня, нам остаётся только переместиться из этого ребёнка в корень и мы получим требуемый в задаче цикл.

Задача F. Ларёк и две куклы

Автор задачи: Егор Юлин, разработчик: Иван Пашков

Для начала заметим, что минимальная величина, на которую мы можем изменить количество проклятий в кукле, равна $\texttt{gcd}(c, m)$ и что максимальная сумма проклятий в куклах достигается или при максимальном количестве проклятий в деревянной кукле, или при максимальном количестве проклятий в тряпичной кукле.

Чтобы понять, какое максимальное количество проклятий может быть у некоторой куклы, для ее начальной стоимости k вычислим $k \bmod \gcd(c, m)$. Тогда максимально возможная стоимость для этой игрушки это $m - \gcd(c, m) + (k \bmod \gcd(c, m))$.

Чтобы узнать количество проклятий другой куклы при определённом количестве проклятий в рассматриваемой, заметим, что если количество проклятий в одной кукле изменилось на величину $s \bmod m$, то у другой — на $(m - s) \bmod m$.

Задача G. Подарок Уэнсдей

Автор задачи и разработчик: Олег Сметанкин

При рассмотрении второго примера из условия можно понять, что простое жадное решение не будет работать. Поэтому для решения задачи используем динамическое программирование. Начнем с базовой идеи и оптимизируем ее. Можно сделать динамику, в которой мы последовательно идём по строке s и решаем к какой из строк приписать текущий символ. Тогда сделаем состояние ДП из трех значений: количество рассмотренных символов в строке s (c_s), количество символов, которые были дописаны к t_1 (c_{t1}) и количество символов, которые были дописаны к t_2 (c_{t2}). Асимптотика решения будет $O(n \cdot |t_1| \cdot |t_2|)$ по времени и памяти, что слишком много.

Заметим, что можно избавиться от последнего параметра, потому что $c_{t2} = c_s - c_{t1}$. Тогда асимптотика получится $O(n \cdot |t_1|)$ по времени и памяти. По времени решение уже проходит, но нужно уменьшить асимптотику по памяти. Так как по строке s мы идем последовательно, нам важны лишь состояния на текущем и прошлом рассмотренных символах для обновления ДП. Поэтому первый параметр c_s можно сделать неявным, для этого просто храним состояния на прошлом символе s и через них обновляем состояния на текущем символе. После полного пересчета для текущего символа можно записать текущее состояние в прошлое. Получится асимптотика $O(n \cdot |t_1|)$ по времени и $O(|t_1|)$ по памяти.

Задача H. Ужасающий эксперимент Уэнсдей

Автор задачи: Александр Гордеев, разработчик: Антон Вдовин

Давайте отсортируем массив и выберем среднее значение из тройки. Тогда нужно выбрать еще по одному элементу слева и справа от текущего. Можно показать, что эти числа должны быть как можно ближе друг к другу. Следовательно, достаточно проверить все подряд идущие тройки в отсортированном массиве.

Задача I. Проблемы бритья

Автор задачи и разработчик: Михаил Ермольев

Эту задачу можно решать по разному, но мы приведём одна из простейших решений. Простой проход по всем клеткам с проверкой может ли она быть покрыта.

Пусть у каждой клетки будет три состояния «Точно нельзя покрыть» — *Blocked*, «Возможно можно покрыть» — *InProgress*, «Точно можно покрыть» — *Done*. Чтобы не усложнять, давайте отдельно проверять можно ли побрить клетку вертикально или горизонтально. Разберём горизонтальную проверку, вертикальная аналогична.

Условимся, что над нашим массивом есть строка полностью заполненная *Blocked*. Давайте обрабатывать строку j. Будем пробегаться по строке, помечая все клетки которые нельзя брить как *Blocked*, а те которые можно брить как *InProgress* и считать количество подряд идущих клеток в переменной *count*. Как только мы дойдём до *Blocked* клетки или до конца строки, проверяем $k \leqslant count$.

Если это так, то пробегаемся назад на *count* клеток и проверяем клетки строки $j - 1$ там мы считаем клетки типа *Done* и *InProgress* в *pred_count* и для всех отрезков $k \leqslant pred\_count$ ставим в линиях j и $j + 1$ *Done*, иначе в линии $j - 1$ заменяем в этом отрезке *InProgress* на *Blocked*.

В случае, когда $k > count$. Пробегаемся по строкам j и $j - 1$ и заменяем все *InProgress* на *Blocked*.

Потом делаем аналогичное по вертикали и в конце проверяем, что каждая нужная по условию клетка покрыта хотя бы в одной матрице.

Данное решение не является оптимальным, однако кажется самым простым.

Задача J. LOIS

Автор задачи и разработчик: Павел Скобелин

Для начала заметим, что если в массиве a больше одного различного элемента, то подойдет любая его перестановка, которая его поменяет.

Далее считаем, что массив состоит из чисел $= x$.

Тогда рассмотрим 5 случаев:

1. x — составное. Тогда представим $x = a \cdot b$, где $1 < a, b < x$. Очевидно, что $a = \frac{x}{b} \leqslant \frac{x}{2}$, аналогично $b \leqslant \frac{x}{2}$. Значит, $a + b \leqslant x$. Тогда давайте любой из элементов x в массиве заменим на a, b, и $x - a \cdot b$ единиц. Понятно, что в таком случае и сумма, и произведение массива не изменится, а числа в массиве только уменьшатся, поэтому не превысят k.

2. $x = 0$. Давайте добавим еще один 0 в конец массива и выполним условие задачи.

3. $x = 1$. Несложно показать, что в таком случае все элементы массива должны быть 1, чтобы произведение массива осталось прежним. Аналогично, количество элементов массива должно остаться прежним, чтобы сохранить сумму. Значит, ответа в таком случае не существует.

4. $x = 2$. Если $n = 1$, то ответа нет. Иначе заменим 2 двойки на одну четверку (предварительно проверив, что $k \geqslant 4$).

5. $x > 2$ — простое. Изначальная сумма в массиве — $n \cdot x$, произведение — x^n. Попробуем преобразовать наш массив. Какие преобразования с ним мы можем делать? Первое — это добавить 1 в наш массив, не меняя произведение, но **строго увеличивая** сумму. Второе — объединить x^i и x^j в одно число x^{i+j}, не меняя произведение, но **строго увеличивая** сумму (при $x > 2$ можно показать, что $x^{i+j} > x^i + x^j$). Таким образом, любое изменение массива либо строго увеличивает сумму, либо строго увеличивает произведение — значит, в таком случае ответа не существует.

Задача K. Инциденты в Неверморе

Автор задачи и разработчик: Павел Скобелин

Будем решать задачу в `offline`. Для начала отсортируем запросы по увеличению умения работать в команде. Заметим, что если какой-то из обитателей имеет умение работать в команде меньшее, чем нужно — в таком случае он не сможет помочь ни в текущем запросе, ни в одном из следующих. Значит, алгоритм такой: пока умение работать в команде у i-го обитателя меньше, чем нужно в запросе, давайте увеличивать умение его работать в команде с помощью особого трюка. Несложно показать, что суммарно таких изменений произойдет не очень много: ведь если $c_i > 1$, то таких изменений произойдет не более $\log M$, а если $c_i = 1$, человек по сути имеет бесконечный скилл работы в команде.

Тогда давайте поддерживать текущие умы обитателей в дереве отрезков: в точке x_i будет содержаться текущий ум человека номер i. Тогда, отвечать на запросы можно с помощью бинарного поиска по ответу: нужно проверить, что сумма умов обитателей на отрезке $[y_j - mid; y_j + mid]$ хотя бы s_j.

Можно реализовать этот алгоритм с помощью неявного дерева отрезков. Но, если воспользоваться техникой сжатия координат, то невность не пригодится — и можно будет даже использовать дерево Фенвика для более быстрого ответа на запросы, но для полного решения задачи это не требовалось.

Итоговая асимптотика $\mathcal{O}(n \log M \log n + q \log^2 n)$

Задача L. Мистический эксперимент Пагсли

Автор задачи: Егор Юлин, разработчик: Иван Пашков

Заметим, что если при массиве $a_1, a_2, \ldots, a_n$ для некоторого элемента a_i выполняется, что $\texttt{xor}(a_1, a_2, \ldots, a_{i-1})$ равняется $\texttt{xor}(a_{i+1}, a_{i+2}, \ldots, a_n)$, то суммарный $\texttt{xor}(a_1, a_2, \ldots, a_n)$ массива равняется a_i (потому что **xor** префикса равняется **xor** суффикса, а значит их суммарный xor равен 0 по свойству **xor**).

Таким образом, чтобы вычислить ответ для текущего массива, нужно понять, сколько элементов в массиве в точности равняется xor всего массива.

Для этого будем поддерживать количество элементов для каждого значения с помощью какого-нибудь словаря, например, с помощью `std::unordered_map<int, int>` в C++ или `java.util.HashMap<Integer, Integer>` в Java.

Будем также поддерживать суммарный **xor** массива. Как пересчитывать? Пусть до замены элемента суммарный **xor** элементов равняется `sum`, старое значение элемента равняется `old_value`, новое равняется `new_value`. При замене элемента новый суммарный **xor** массива равняется $\texttt{sum} \oplus \texttt{old\_value} \oplus \texttt{new\_value}$.

Итоговый алгоритм:

1. В самом начале инициализируем словарь всеми значениями и считаем суммарный **xor** массива.
2. При запросе первого типа пересчитываем суммарный **xor** массива и обновляем данные в словаре;
3. При запросе второго типа получаем из словаря и выводим количество значений, которые равняются суммарному **xor** всех элементов.