

Задача А. Мысли перед сном

Разработчик задачи: Александр Малькевич

В этой задаче ответ равен числу $x - 2019 + 1 = x - 2018$.

Задача В. Как можно больше

Разработчик задачи: Софья Тарасова

Если $a > b$, поменяем их местами: на ответ это не влияет, ведь Вова сам выбирает направление каждого разлома. Далее считаем, что $a < b$. Нам нужно рассмотреть два случая:

1. Пусть оба разлома идут по рядам (случай, когда оба идут по столбцам, аналогичен). Линии после рядов a и b делят шоколадку на три полосы высотой a , $b - a$ и $n - b$, и каждая полоса имеет ширину n . Количество долек в самом большом куске будет равно $v_1 = n \cdot \max(a, b - a, n - b)$.
2. Пусть разлом с номером a идёт по ряду, а с номером b — по столбцу (случай, когда a идёт по столбцу, а b по ряду, аналогичен). Тогда шоколадка распадается на четыре прямоугольника со сторонами a или $n - a$ по вертикали и b или $n - b$ по горизонтали. Самый большой из них получается, если взять большую часть по каждому направлению: $v_2 = \max(a, n - a) \cdot \max(b, n - b)$.

Ответ равен максимальному из двух найденных чисел, то есть — $\max(v_1, v_2)$. Решение работает за $\mathcal{O}(1)$.

Задача С. Итоговая оценка

Автор задачи: Алексей Егоров. Разработчик задачи: Александр Чугунов

В этой задаче решение состоит из двух частей: сначала найдем оценку за тестовую часть, затем посчитаем итоговую оценку по правилу округления.

1. Обозначим оценку за тест через $test$. Доля верных ответов равна $\frac{x}{k}$. Условие «доля не меньше 90%» можно записать так:

$$\frac{x}{k} \geq \frac{90}{100}.$$

Так как $k > 0$, обе части можно умножить на $100k$ и получить равносильное условие без деления:

$$100 \cdot x \geq 90 \cdot k.$$

Остальные пороги записываются так же. Проверяем их сверху вниз, и первый выполненный задает оценку:

- если $100x \geq 90k$, то $test = 5$;
- иначе если $100x \geq 80k$, то $test = 4$;
- иначе если $100x \geq 70k$, то $test = 3$;
- иначе если $100x \geq 60k$, то $test = 2$;
- иначе $test = 1$.

Порядок проверки важен: например, доля 95% удовлетворяет и условию для 90%, и условию для 60%, но оценку задает первое выполненное, то есть самое высокое.

2. Определение итоговой оценки. Если сумма $test + a$ является чётным числом, то оценка определяется однозначно: она будет равна $\frac{test + a}{2}$. Иначе нам нужно будет воспользоваться правилами округления.

Если $a > test$, то округление будет в большую сторону (в сторону задачи), иначе в меньшую. В первом случае ответ будет равен $\frac{test + a + 1}{2}$, а во втором — $\frac{test + a - 1}{2}$.

Решение можно реализовать за $\mathcal{O}(1)$.

Задача D. Квадратные цифры

Разработчик задачи: Максим Сименков

Решение можно реализовать следующим образом:

1. Берём последнюю цифру числа x : $x \bmod 10$.
2. После этого к ответу прибавляем её квадрат, умноженный на текущую степень десятки (вес разряда).
3. Удаляем последнюю цифру из числа, то есть делим x целочисленно на 10.

Алгоритм выше нужно повторять до тех пор, пока мы не рассмотрим все цифры числа, то есть пока x будет больше 0. При данных ограничениях решение будет работать за $\mathcal{O}(1)$.

Задача E. Необычная сортировка

Разработчик задачи: Анастасия Корюкина

Это стандартная задача на сортировку, в которой нужно лишь реализовать функцию сравнения двух чисел (компаратор). Далее можно будет воспользоваться либо встроенной сортировкой, либо реализовать её самостоятельно.

Чтобы реализовать функцию сравнения двух чисел, нужно у каждого числа научиться находить старший разряд. Это можно сделать разными способами, самый популярный — удалять последнюю цифру числа (делить число целочисленно на 10) до тех пор, пока число не станет однозначным.

Задача F. Перестановки в матрице

Разработчик задачи: Мария Потапова

Данные ограничения позволяют наивно менять местами строку и столбец для каждой операции за $\mathcal{O}(n)$. Суммарно решение можно реализовать за $\mathcal{O}(q \cdot n)$.

Задача G. Напиши меня

Автор задачи: Екатерина Шилыева. Разработчик задачи: Мария Ткаченко

Сохраним каждую букву как пять строк.

Будем строить ответ построчно. Для каждой из пяти строк ответа пройдем по всем буквам строки s , для каждой выведем нужную строчку. Между соседними буквами добавим одну точку. Перед первой и после последней буквы добавлять точки не нужно.

Решение можно реализовать за $\mathcal{O}(n)$.

Задача H. Максимальный палиндром

Автор задачи: Екатерина Шилыева. Разработчик задачи: Тимофей Венедиктов

Палиндром читается одинаково слева направо и справа налево, поэтому его можно описать так: левая половина + (возможно) одна средняя буква + левая половина в обратном порядке. Значит, каждая буква, кроме средней, должна встречаться в палиндроме чётное число раз: поровну в левой и правой половине.

Как получить максимальную длину? Посчитаем, сколько раз каждая буква встречается в строке s . Пусть буква встречается c раз. Из неё можно составить $\lfloor c/2 \rfloor$ пар. Пары берём все, ведь каждая пара удлиняет палиндром на 2. Если c нечётно, то одна буква остаётся без пары. Её можно поставить в середину. В середину можно поставить только одну букву, поэтому если лишних букв несколько, использовать получится только одну из них. Итого длина ответа равна $2 \cdot \sum \lfloor c/2 \rfloor$, плюс 1, если хотя бы одна буква встречается нечётное число раз.

Чтобы добиться лексикографической минимальности, — будем набирать буквы в ответ по порядку: — сначала $\lfloor c[a]/2 \rfloor$ букв a , потом $\lfloor c[b]/2 \rfloor$ букв b и так далее. А в середину возьмём просто минимальную букву, количество которой нечётно.

Решение можно реализовать за $\mathcal{O}(n)$.

Задача I. Слагаемые

Разработчик задачи: Владимир Аверенков

Сначала отсортируем массив. Перестановка элементов не меняет ответ: каждому элементу по-прежнему соответствует отдельная позиция, а порядок индексов в условии не ограничен.

Зафиксируем позицию i и обозначим $x = a_i$. Удалим из отсортированного массива только элемент на этой позиции. Получим отсортированный массив b длины $m = n - 1$.

Теперь нужно найти в b две различные позиции l и r , для которых $b_r - b_l = x$. Так как $x \geq 0$, достаточно рассматривать $l < r$. При $x = 0$ мы ищем два одинаковых элемента на разных позициях.

Далее задачу можно решить с помощью техники двух указателей. В нумерации с нуля изначально $l = 0$, $r = 1$. Пока $r < m$, сравниваем $d = b_r - b_l$ с x :

- Если $d = x$, индекс i подходит. Увеличиваем ответ на один и переходим к следующему i .
- Если $d < x$, увеличиваем r .
- Если $d > x$, увеличиваем l . Если указатели совпали, также увеличиваем r , чтобы сохранить $l < r$.

Если правый указатель вышел за границу массива, подходящей пары нет. При $n < 3$ цикл поиска не выполняется, поэтому ответ равен нулю.

Оценим время работы. Сортировка занимает $\mathcal{O}(n \log n)$ времени. Для каждой позиции построение массива b и поиск двумя указателями занимают $\mathcal{O}(n)$ времени, так как каждый указатель только увеличивается. Общая сложность решения равна $\mathcal{O}(n^2)$ по времени.

Задача J. Делись или будь делимым

Разработчик задачи: Егор Зарубин

Делимость — это отношение порядка, а значит, если a делит b , а b делит c , то a делит c . Также необходимым условием того, что a делит b , является $a \leq b$.

Давайте отсортируем данный массив по возрастанию. Проверим, что для любого $i = 1 \dots n - 1$ выполнено, что a_{i+1} делится на a_i без остатка (в этом случае выводим YES, иначе NO).

Докажем, что решение выше корректно. Если все элементы массива хорошие, то для любой пары должно выполняться хотя бы одно из условий делимости, в частности, это верно и для соседних элементов. Если же для некоторого i число a_i не делит a_{i+1} , то a_{i+1} не делит a_i , так как из-за сортировки $a_{i+1} > a_i$. Значит, эти два числа не делятся друг на друга, и массив содержит нехорошие элементы. С другой стороны, если все соседние пары делятся, то по транзитивности любой элемент делит все элементы правее него, поэтому все числа хорошие.

Задача K. Хоккей

Автор и разработчик задачи: Александр Волков

Нужно эффективно находить простые числа на отрезке от 2 до $d \cdot w$; для этого будем использовать решето Эратосфена. Чтобы проверить, есть ли тренировка в день, когда Женя хочет на неё сходить, будем брать остаток от деления соответствующего простого числа на d и проверять, есть ли в этот день тренировка. Для последнего действия можно завести множество или булевый список длины d , содержащий дни тренировок.

Задача L. Сапёр 4 x 4

Автор и разработчик задачи: Екатерина Шилыева

Заметим, что поле для игры достаточно маленькое, и можно перебрать все возможные варианты расположения бомб. Будем использовать рекурсивный перебор для учёта всех вариантов. В конце, после того как в переборе получено какое-то расположение бомб на поле, нужно сверить конфигурацию с известными цифрами на поле. Так как про каждую клетку на момент окончания одной ветки перебора известно, пустая она или есть ли в ней бомба, можем посчитать количество бомб в окрестности каждой клетки с цифрой и проверить, равно ли оно самой цифре. Если все цифры и количество бомб совпадают, этого достаточно, чтобы определить, что перебранная конфигурация корректна.

Чтобы теперь определить, где точно стоит бомба, а где нет, посмотрим на все найденные корректные конфигурации. Если в клетке во всех таких конфигурациях стоит бомба, то в клетке точно бомба и следует вывести 'B'. Если в клетке во всех таких конфигурациях нет бомбы, то клетка точно безопасна и следует вывести 'S'. Иначе, если в клетке в каких-то конфигурациях возможна бомба, а в каких-то возможно её отсутствие, про клетку нельзя ничего однозначно сказать, и следует вывести '?'.

Задача М. Восстановление массива

Автор задачи: Софья Тарасова. Разработчик задачи: Михаил Грибакин

Удобно представить условия задачи в виде графа.

Каждому элементу массива a_i сопоставим вершину i . Каждому утверждению $a_x + a_y = s$ сопоставим ребро между вершинами x и y с весом s .

Рассмотрим одну компоненту связности этого графа. Пусть значение некоторой стартовой вершины равно неизвестному числу t .

Тогда по ребру (v, u) с весом s выполняется $a_u = s - a_v$.

Следовательно, значение каждой вершины компоненты можно представить в виде $a_v = t + shift_v$ или $a_v = -t + shift_v$.

Знак перед t меняется при каждом переходе по ребру. Его удобно хранить при помощи двух цветов вершин.

Во время DFS будем хранить:

- $color_v$ — знак перед неизвестным t ;
- $shift_v$ — свободную часть выражения для a_v .

Для стартовой вершины положим $a_{start} = t$, то есть $shift_{start} = 0$.

Если по ребру с весом s мы переходим из вершины v в ещё не посещённую вершину u , то $color_u = 1 - color_v$ и $shift_u = s - shift_v$.

Теперь рассмотрим ребро, которое ведёт в уже посещённую вершину.

Если цвета концов ребра различны, коэффициенты при t имеют разные знаки и взаимно уничтожаются. Такое ребро не позволяет определить t .

Если же цвета концов совпадают, то коэффициенты при t одинаковы. Например, если $a_v = t + shift_v$ и $a_u = t + shift_u$, то из условия $a_v + a_u = s$ получаем $2t + shift_v + shift_u = s$, откуда $t = \frac{s - shift_v - shift_u}{2}$.

Аналогично обрабатывается случай, когда обе вершины имеют коэффициент $-t$.

Таким образом, если в компоненте существует ребро между вершинами одного цвета, значение t определяется однозначно. После этого однозначно определяются значения всех вершин этой компоненты.

Если такого ребра нет, компонента является двудольной. Тогда t остаётся свободным параметром, поэтому ни одно значение в этой компоненте нельзя восстановить однозначно.

Иными словами:

- если компонента графа двудольная, ни один её элемент не определяется однозначно;
- если компонента не двудольная, все её элементы определяются однозначно.

После обхода каждой компоненты, если значение t удалось определить, восстанавливаем все её элементы по формулам $a_v = t + shift_v$ или $a_v = -t + shift_v$.

Для хранения графа достаточно в каждом ребре хранить номер соседней вершины и сумму s .

Каждое ребро и каждая вершина обрабатываются константное число раз, поэтому обход графа работает за $\mathcal{O}(n + m)$.

Если восстановленные индексы дополнительно сортировать перед выводом, итоговая асимптотика будет $\mathcal{O}(n + m + k \log k)$, где k — количество однозначно восстановленных элементов.

Можно также избежать сортировки и после обработки всех компонент пройти по индексам от 1 до n , выводя только восстановленные элементы. Тогда итоговая асимптотика останется $\mathcal{O}(n + m)$.

Так как промежуточные значения $shift_v$ могут достигать величин порядка $m \cdot s$, для всех вычислений значений массива необходимо использовать 64-битный целый тип, например `long long`.

Задача N. Душная библиотека

Автор и разработчик задачи: Илья Богатырев

Заметим, что если в некоторый день книга выдаётся читателю, то это либо первый день, либо в тот же день ранее какая-то книга была возвращена в библиотеку: либо та же самая (и книга выдаётся тому, кто ждал её), либо какая-то другая (читатель сдал предыдущую книгу и получает следующую по списку). Поэтому после первого дня будем рассматривать только дни, в которые какие-то книги сдаются.

Будем поддерживать информацию обо всех книгах, которые сейчас выданы читателям, с возможностью эффективного получения информации о ближайшем возврате. Например, можно использовать `set` или `priority_queue` в C++ или `heapq` в python (ключ — пара из номера дня и номера читателя). Будем каждый раз извлекать информацию обо всех возвратах, которые происходят в ближайший день. Для каждой возвращённой книги обновим количество экземпляров в наличии, а для каждого из читателей найдём следующую по списку книгу и запомним, что он хочет эту книгу получить. Удобно для каждого читателя поддерживать текущую позицию в списке чтения, чтобы по номеру читателя всегда иметь возможность определить номер предыдущей и следующей выдаваемых книг.

Чтобы обрабатывать все выдачи в порядке возрастания номеров, для каждой книги будем поддерживать информацию обо всех читателях, которые хотят её получить. Поскольку выдавать каждый очередной экземпляр нужно читателю с минимальным номером, то хранить всех ожидающих также можно в `set`, `priority_queue` или `heapq` (ключ — номер человека). Каждый раз при выдаче уменьшим число экземпляров в наличии и создадим новое событие возврата соответствующей книги в будущем. День возврата известен, так как известны текущий день и количество экземпляров книги в момент выдачи.

Общая временная сложность моделирования — $\mathcal{O}(s \log n)$, где s — сумма всех s_i и по условию не превосходит 10^5 .

Важно заметить, что нельзя в каждый из рассматриваемых дней перебирать все книги (это приведёт к временной сложности $\mathcal{O}(sm)$ или хуже). По итогам утренней сдачи книг нужно запомнить только те, которые были сданы, либо являются очередными книгами в списке кого-то из читателей, возвративших другие книги. Затем при выдаче книг нужно обрабатывать только их.

Задача O. Уравняй НОД массивов

Разработчик задачи: Никита Насонков

Если изначально $\gcd(a) = \gcd(b)$, то ответ равен 0.

За две операции всегда можно добиться равенства НОД: достаточно увеличить a_1 и b_1 до простого числа $10^9 + 7$. Все остальные элементы меньше этого числа, поэтому НОД каждого массива станет равен 1.

Остаётся проверить, можно ли обойтись одной операцией. Пусть мы изменяем массив b , а $g = \gcd(a)$. В массиве b должно быть не более одного элемента, не делящегося на g : иначе хотя бы один такой элемент останется неизменным, и НОД массива не будет равен g .

Если такой элемент ровно один, выберем его; если все элементы делятся на g , выберем любую позицию. Пусть выбран индекс i , а h — НОД остальных элементов. Заменим b_i на

$$y = \left(\left\lfloor \frac{b_i}{h} \right\rfloor + 1 \right) h + g.$$

Тогда $y > b_i$, а $\gcd(h, y) = \gcd(h, y \bmod h) = \gcd(h, g) = g$. Прибавление не превышает $h + g \leq 2 \cdot 10^9$, поэтому операция допустима.

Аналогично проверим возможность изменения массива a за одну операцию. Если обе проверки неуспешны, выведем конструкцию за две операции.

Время работы — $\mathcal{O}((n + m) \log A)$, где $A = 10^9$ ($\log A$ из-за нахождения НОД алгоритмом Евклида). Память — $\mathcal{O}(n + m)$ для хранения массивов и $\mathcal{O}(1)$ дополнительно.

Задача Р. JoJo's Bizarre Subsequence

Автор задачи: Иван Добрынин. Разработчик задачи: Александр Степаненко

Будем использовать динамическое программирование и дерево отрезков. Пусть dp_i — максимальная длина подходящей подпоследовательности, заканчивающейся в позиции i . Тогда $dp_i = 1 + \max dp_j$ по всем $j < i$, для которых $a_j < a_i$ и $c_j \neq c_i$. Если таких j нет, то $dp_i = 1$. Построим дерево отрезков по значениям a_i . В каждой вершине будем хранить две пары (dp, c) :

- Максимальное значение dp и его цвет.
- Максимальное значение dp среди элементов другого цвета.

Этого достаточно, поскольку при запросе мы исключаем только один цвет. Если цвет первого максимума совпадает с c_i , берём второй, иначе — первый.

Будем обрабатывать элементы слева направо. Для каждого элемента i сделаем запрос на отрезке $[1, a_i - 1]$, чтобы найти максимальное dp_j с цветом, отличным от c_i . После вычисления dp_i обновим позицию a_i в дереве отрезков. При объединении двух вершин достаточно выбрать две лучшие пары с различными цветами среди четырёх кандидатов. Ответом будет максимальное значение dp_i среди всех элементов.

Такое решение работает за $\mathcal{O}(n \log n)$.