Задача A. Наиболее вкусный торт

Подсказка 1. Предположим, вы хотите выбрать куски пирога i и j. Сможете ли вы сделать их подряд идущими за 1 ход?

Подсказка 2. Ответом на задачу является сумма двух максимальных весов.

Решение. В самом деле, ответом на задачу является сумма двух максимальных весов.

Вы всегда можете выбрать два максимальных по весу куска. Допустим, это куски a_i и a_j $(i < j)$, вы можете развернуть участок $[i, j-1]$, чтобы сделать их подряд идущими.

Очевидно, результат не может превышать сумму двух максимальных весов. Следовательно, такое решение даёт искомый максимальный ответ.

Достаточно перебрать все пары кусков пирога, чтобы получить полное решение по задаче. Однако, эта задача может быть решена за время $O(n \log n)$ путём сортировки и вывода двух суммы последних элементов. Более того, задача может быть и решена за $O(n)$. Для этого достаточно найти два максимальных значения за линейное время (например, в один или два прохода по массиву).

Асимптотика: $O(t \cdot n^2)$, $O(t \cdot n \log n)$ или $O(t \cdot n)$.

Задача B. Префиксные удаления

Подсказка 1. Есть ли символы, которые никогда нельзя удалять?

Подсказка 2. Вы не можете удалить самое правое вхождение каждого символа. Можно ли удалить другие вхождения?

Подсказка 3. Если вы можете удалить $s_1, s_2, \ldots, s_{i-1}$ и s_i **не является** крайним правым вхождением символа, вы также можете удалить s_i.

Решение. Пусть a — начальная строка. Для произвольной строки z определим $z(l, r) = z_l z_{l+1} \ldots z_r$, т. е. подстроку строки z от l до r включительно. Финальное состояние строки — это $a(k, n)$ для некоторого k.

В финальной строке $x = 0$, поэтому первый символ больше нигде не встречается в s. Это означает, что $a_k \neq a_{k+1}, a_{k+2}, \ldots, a_n$. Другими словами, a_k — это самое правое вхождение символа в s.

Можно ли удалить a_i, если $a_i \neq a_{i+1}, a_{i+2}, \ldots, a_n$? Для этого нужно удалить $a(l, r)$ $(l \leqslant i \leqslant r)$: это означает, что должно существовать $a(l', r') = a(l, r)$ для некоторого $l' > l$. Итак, $a_{i+l'-l} = a_i$, то есть a_i не является крайним правым вхождением символа, противоречие.

Следовательно, k — наименьший индекс, для которого $a_k \neq a_{k+1}, a_{k+2}, \ldots, a_n$.

Можно найти k, перебирая символы в строке **справа налево** и поддерживая счетчики количества вхождений каждого символа. В самом деле, $a_i \neq a_{i+1}, a_{i+2}, \ldots, a_n$ тогда и только тогда, когда количество вхождений символа a_i равно 0. Искомое значение k — это минимальный подходящий индекс i.

Асимптотика: $O(n)$.

Задача C. Алиса и торт

Подсказка 1. Можете ли вы найти начальный вес торта, с помощью которого можно получить массив a?

Подсказка 2. Начальный вес торта равен сумме весов кусочков. Давайте просимулируем разрезание, начиная с целого торта. Как выбрать, какой кусочек разрезать следующим?

Подсказка 3. Если вес самого большого кусочка не встречается в a, то этот кусочек нужно разрезать. Как быстро находить вес самого большого кусочка?

Подсказка 4. Попробуйте хранить a и ваш не полностью разрезанный торт в двух очередях с приоритетом.

Решение. Для начала, попробуем найти начальный вес торта. Когда разрезается очередной кусочек торта веса w, сумма весов образовавшихся кусочков равна $\lfloor \frac{w}{2} \rfloor + \lceil \frac{w}{2} \rceil$:

- если w чётно, то $\lfloor \frac{w}{2} \rfloor + \lceil \frac{w}{2} \rceil = \frac{w}{2} + \frac{w}{2} = w$;
- если w нечётно, то $\lfloor \frac{w}{2} \rfloor + \lceil \frac{w}{2} \rceil = \frac{w-1}{2} + \frac{w+1}{2} = w$.

Таким образом, сумма весов не меняется, а исходный вес торта равен сумме весов его кусочков.

Давайте начнём с торта b веса $b_1 = \sum_{i=1}^{n} a_i$, разрежем его на кусочки весов b_i и попробуем сделать их равными a. В любой момент времени имеет смысл рассматривать только наибольший кусочек b_i, так как можно однозначно определить, нужно ли его разрезать или нет, а именно:

- если b_i не содержится в a, то его нужно разрезать;
- если $b_i = a_j$ для какого-то j, то возможно сопоставить a_j только либо с b_i, либо с таким b_k, что $b_k = a_j = b_i$ (так как кусочков больших, чем b_k, нет): это эквивалентно удалению a_j и b_i из a и b соответственно.

Также стоит заметить, что если в какой-то момент максимальный элемент b стал меньше максимального элемента a, то ответ `NO`.

Если хранить a и b в любой структуре данных, которая поддерживает вставку числа, запрос максимума и удаление максимума (к примеру, очередь с приоритетом или `std::multiset` из C++), то будет работать следующий алгоритм.

Пока a и b оба не пусты,

- если максимальный элемент b меньше максимального элемента a, то выводим `NO` и выходим из цикла;
- если максимальный элемент b равен максимальному элементу a, то удаляем их оба из a и b;
- если максимальный элемент b больше максимального элемента a, то удаляем его из b и разрезаем (то есть, вставляем $\lfloor \frac{b_{max}}{2} \rfloor$ и $\lceil \frac{b_{max}}{2} \rceil$ в b).

Если a и b оба пусты в конце, то выводим `YES`, иначе `NO`.

Асимптотика решения: $O(n \log n)$.

Задача D. Урок зельеварения

Подсказка 1. $n - 1$ факт образуют дерево (в терминах теории графов).

Подсказка 2. Предположим, что объем ингредиента 1 равен 1. Для каждого i объем ингредиента i будет равен c_i/d_i для некоторых целых чисел $c_i, d_i > 0$. Как сделать объем каждого ингредиента целым значением?

Подсказка 3. Оптимальный объем ингредиента 1 равен $\operatorname{lcm}(d_1, d_2, \ldots, d_n)$.

Подсказка 4. Можно ли найти показатель степени каждого простого числа $p \leqslant n$ в $\operatorname{lcm}(d_1, d_2, \ldots, d_n)$?

Подсказка 5. Если вы посещаете вершины в порядке обхода в глубину, каждое ребро изменяет показатель степени $O(\log n)$ простых чисел. В каждый момент поддерживайте факторизацию c_i/d_i для текущей вершины, т. е. показатели степени каждого $p \leqslant n$ (они могут быть отрицательными). Для каждого p храните минимальный показатель степени среди всех вершин.

Решение. Построим граф в котором вершины — это ингредиенты, а ребра — факты отношений. Запустим обход в глубину, начиная с вершины 1. Будем поддерживать массив f таким образом, чтобы f_p было показателем степени p в объеме ингредиентов текущей вершины. Также будем поддерживать $v_i = \frac{c_i}{d_i} \bmod 998\,244\,353$. В начале объем ингредиентов (вершины 1) равен 1, поэтому $f_p = 0$ для каждого p. Всякий раз, когда вы переходите от вершины i к вершине j, и $r_i/r_j = x/y$, нужно:

- для каждого p^k такого, что $p^k \mid x$ и $p^{k+1} \nmid x$, уменьшить f_p на 1;
- для каждого p^k, такого что $p^k \mid y$ и $p^{k+1} \nmid y$, увеличить f_p на 1.

Обратите внимание, что существует $O(\log n)$ таких значений p^k для каждого ребра, и вы можете найти их, предварительно вычислив либо наименьший простой множитель (с помощью решета Эратосфена), либо всю факторизацию каждого целого числа на отрезке $[2, n]$.

Пусть $g_p \geqslant 0$ — минимальное значение f_p за время работы поиска в глубину. Для каждого p нужно умножить объем всех ингредиентов на p^{-g_p}, чтобы объем в каждой вершине стал целочисленным.

Ответ — сумма v_i, умноженная на все значения p^{-g_p}. Обратите внимание, что вычисления требовалось производить по модулю 998 244 353.

Временная асимптотика: $O(n \log n)$.

Задача E. Арифметические операции

Подсказка 1. Рассмотрим каждый элемент массива как точку на плоскости с координатами (i, a_i). Тогда все элементы, которые **не будут затронуты** операциями, лежат на одной прямой. Найдите прямую, на которой максимальное количество точек.

Подсказка 2. Пусть m — верхняя граница a_i. Предполагаемая временная сложность $O(n\sqrt{m})$.

Подсказка 3. Пусть m — верхняя граница a_i. Пусть d — предполагаемый шаг итоговой арифметической прогрессии. Решите задачу по-отдельности для случаев, когда $|d| < \sqrt{m}$ и $|d| \geqslant \sqrt{m}$.

Решение. Как объяснено в подсказках, вместо вычисления наименьшего количества операций мы будем вычислять наибольшее количество элементов, которые не будут затронуты, и решим задачу по-отдельности для случаев $|d| < \sqrt{m}$ и $|d| \geqslant \sqrt{m}$. Временная сложность составит $O(n\sqrt{m})$, где m — верхняя граница a_i.

Пусть d — шаг итоговой арифметической прогрессии. Будем предполагать, что $d \geqslant 0$, так как задача для отрицательного d решается разворотом массива и применением решения для неотрицательного случая. Если d фиксировано заранее, мы можем решить задачу за $O(n)$, группируя элементы по значению $a_i - d \cdot i$, одинаковому для всех чисел на одной прямой. Ответом на задачу будет n минус размер самой большой получившейся группы.

Для $d < \sqrt{m}$ мы можем использовать описанный выше алгоритм для обработки всех возможных d за время $O(n\sqrt{m})$. Мы можем хранить хэш-таблицу из индекса группы в количество элементов в группе, или мы можем просто хранить массив, поскольку индексы группы имеют диапазон не более $O(n\sqrt{m})$, что можно разместить в пределах ограничения по памяти.

Для $d \geqslant \sqrt{m}$ заметим, что если у нас есть два индекса i, j такие, что $j > i + \sqrt{m}$, то хотя бы над одним из них обязательно должна быть выполнена операция, потому что разница между ними должна быть $a_j - a_i \geqslant \sqrt{m} \cdot d > m$, что невозможно. Другими словами, если мы рассмотрим множество элементов, которые не изменяются операцией (лежат на одной прямой), то все эти элементы лежат в пределах одного отрезка длиной не более $\sqrt{m}$.

Так, мы можем построить граф между индексами с ребрами $i \to j$ и метками x, если $i < j \leqslant i + \sqrt{m}$ и $\frac{a_j - a_i}{j - i} = x$. Этот граф имеет не более чем $n\sqrt{m}$ ребер. Тогда нам нужно найти самый длинный путь в графе, у которого все ребра имеют одинаковую метку. Это можно сделать с помощью динамического программирования — пусть $dp_{i,d}$ — длина самого длинного пути, заканчивающегося индексом i, у которого все ребра имеют метку d. Для каждого i нам нужно проверить ребра до j, где $i - \sqrt{m} < j < i$. Это означает, что временная сложность равна $O(n\sqrt{m})$. Чтобы хранить значения $dp_{i,d}$ разреженно, мы можем использовать хэш-таблицу.

Итоговая временная асимптотика: $O(n\sqrt{m})$.

Задача F. Минимальное XOR-ирование строки

Подсказка 1. Пусть $f(s, x)$ равно строке t такой, что $t_i = s_{i \oplus x}$. Решение не только ищет такое x, что $f(s, x)$ лексикографически минимально, но и конструирует массив из чисел $0 \leqslant x < 2^n$, отсортированных с помощью компаратора $f(s, i) < f(s, j)$.

Подсказка 2. Решение похоже на стандартный метод построения суффиксного массива.

Подсказка 3. Пусть a_k — это массив целых чисел от 0 до $2^n - 1$, отсортированных согласно лексикографическому порядку первых 2^k символов $f(s, x)$. Ответом на задачу является $f(s, a_{n,0})$. Можете ли вы построить массив a_k, зная массив a_{k-1}?

Решение. Базовый случай a_0 строится довольно просто.

Чтобы получить a_k из a_{k-1}, мы сначала построим дополнительный массив целых чисел v, используя a_{k-1}, такой, что $v_i < v_j$ тогда и только тогда, когда $f(s, i)_{0..2^{k-1}} < f(s, j)_{0..2^{k-1}}$. Заметим, что строки длины 2^k, которые нам нужно сравнивать для получения массива a_k, всегда состоят из двух строк длины 2^{k-1}, порядок которых задается массивом v. Это значит, что мы можем отсортировать массив a_k, сравнивая пары чисел: $(v_i, v_{i \oplus 2^{k-1}}) < (v_j, v_{j \oplus 2^{k-1}})$.

Если известен массив a_n, то $f(s, a_{n,0})$ является ответом на задачу. Таким образом, решение работает за $O(2^n n^2)$ времени, что можно соптимизировать до $O(2^n n)$ времени, используя тот факт, что пары чисел из целого интервала $[0, m]$ могут быть отсортированы за $O(m)$ времени с помощью сортировки подсчетом. Последняя оптимизация не являлась обязательной для получения вердикта «Полное решение».

Задача G. Снежная гора

Подсказка 1. Будем говорить, что вершина «разворотная», если у нее есть хотя бы один сосед такой же высоты. Оптимальная стратегия для лыжника состоит в том, чтобы доехать до разворотной вершины с наименьшей высотой, ездить вперед и назад между этой вершиной и ее соседом, пока не будет израсходована вся энергия, а затем доехать до базы.

Подсказка 2. Предположим, что лыжник начинает с вершины v и выбирает вершину u в качестве разворотной, этот лыжник «потеряет» в общей сложности h_u единиц кинетической энергии, проехав по пути общей длиной $2h_v - h_u$. Обратите внимание, что если таких u не существует, то h_v энергии будет потеряно, так как лыжник поедет прямо на базу. Пусть w_v будет этой «потерянной» величиной. Вместо вычисления максимальной длины пути лыжника, начинающегося в вершине v, вычислим w_v. Ответ для этой вершины $2h_v - w_v$.

Подсказка 3. Попробуем решить такую, казалось бы, несвязанную задачу: пусть S — множество разворотных вершин. Каково максимально возможное значение $\sum\limits_{v \in S} h_v$?

Подсказка 4. Наибольшее возможное значение $\sum\limits_{v \in S} h_v$ ограничено $O(n)$. Следовательно, существует не более $O(\sqrt{n})$ различных значений w_v среди всех вершин. Решим задачу для каждого значения w_v по отдельности.

Решение. Для каждого набора разворотных вершин одинаковой высоты мы можем вычислить набор начальных вершин, из которых достижима хотя бы одна из данного множества разворотных вершин. Для этого разобьем граф на слои, содержащие вершины с одинаковой высотой. Пусть c_v — минимальная энергия, необходимая для достижения какой-либо вершины из множества разворотных, начиная в вершине v. c_v можно вычислить с помощью нахождения кратчайших путей, если считать, что ребра в пределах одного слоя имеют длину 1, а ребра из слоя i в $i + 1$ имеют длину -1. Мы можем использовать поиск в ширину, найти кратчайшие пути в одном слое, а затем легко перейти к следующему слою. Делать это будем для $O(\sqrt{n})$ различных начальных высот, поэтому общая временная асимптотика решения составляет $O(n\sqrt{n})$.