,

Задача 1A. Ещё одна n-мерная шоколадка

За A обозначим максимальное значение a_i

Для начала решим задачу за $O(n \cdot k \cdot f(k, A))$ с помощью динамического программирования.

Положим $dp[i][j]$ - максимальный возможный объем наименьшего кусочка, если по первым i измерениям мы разделили шоколадку на j частей. Если мы разделили более чем на k частей, так же положим результат в $dp[i][k]$. В пересчете нам нужно решить на сколько часей делить шоколадку вдоль очередного измерения. Рассмотрим несколько способов это сделать.

- Можно за $O(k)$ перебрать состояние, в которое переходим, и из этого посчитать на сколько частей нужно разделить шоколадку вдоль очередного измерения. - Получим $O(n \cdot k^2)$

- Можно за $O(A)$ перебрать на сколько частей мы делим шоколадку вдоль очередного измерения.

- Находясь в состоянии $dp[i][j]$, можно перебирать b_i - на сколько частей делить шоколадку, пока $j \cdot b_i \leqslant k$. Можно показать, что такое решение будет работать за $O(n \cdot k \cdot \ln k)$

Ключевая идея

- предположим нам нужно разделить шоколадку на 10 частей, и вдоль первых измерений мы уже разделили её на 5 частей, или на 6 частей, или на $7, 8$ или 9 частей. Все эти состояния не различимы для нас, потому что во всех этих случаях нам нужно разделить шоколадку ещё хотя бы на 2 части. Осталось понять сколько всего таких «интересных» состояний и научиться их хранить. Для этого есть несколько подходов, разберем один из них:

- нам интересны все значения $\lceil \frac{k}{i} \rceil$ для $i = 1, 2, \dots k$ - это то, на сколько частей ещё может быть нужно разделить шоколадку. Среди них всего $O(\sqrt{k})$ различных, так как либо $i \leqslant \sqrt{k}$, либо само значение $\lceil \frac{k}{i} \rceil \leqslant \sqrt{k}$. Если сделать все эти числа состояниями, и пересчитываться, перебирая состояние, в которое переходить, получим $O(n \cdot \sqrt{k} \cdot \sqrt{k}) = O(n \cdot k)$ - этого всё ещё не достаточно, чтобы решить полую задачу.

Последнее наблюдение

Если мы находимся в состоянии $dp[i][remain]$ где $remain = \lceil \frac{k}{i} \rceil$ для какого-то i - применим к нему ту же идею. Из него нам интересны переходы в состояния $\lceil \frac{remain}{j} \rceil$ для $j = 1, 2, \dots remain$. Какая асимптотика получится, если перебирать только интересные переходы?

$n \cdot (\sum_{r=1}^{\sqrt{k}} 2 \cdot \sqrt{r} + 2 \cdot \sqrt{\lceil \frac{k}{r} \rceil})$

- можно показать, что это $O(n \cdot k^{3/4})$ - что решает задачу

Задача 1B. Цены на бензин

Для начала поймём, что от нас требуется. Дано дерево, в каждой вершине которого записан диапазон цен для этой вершины. Запрос это пара путей равной длины, цены на i-ых вершинах вдоль этих путей должны быть равны для всех i. Нужно найти количество способов расставить цены на вершинах для каждого префикса ограничений.

Начнём с медленного решения задачи. Будем хранить компоненты связности(в каждой цены на вершинах должны быть равны). Для каждой из них храним допустимый диапазон цен. Ответом будет произведение длин диапазонов по всем компонентам. Будем идти вдоль путей и объединять 2 вершины в одну компоненту с помощью СНМ. Это уже набирает 31 балл. Используем эвристику сжатия путей и ранговую эвристику в СНМ, также обходим пути запросов за линейное время от их длин - 47 баллов. Понятно, что для ускорения этого решения надо быстрее искать моменты, когда две вершины объединяются в одну компоненту.

Сначала разберём долгое решение. Сделаем heavy-light decomposition, с помощью которого будем хешировать пути в дереве, приняв за символ для вершины номер корня её компоненты. Теперь с помощью бинпоиска будем искать первый момент, когда хеши на префиксах путей различаются, то есть две вершины объединяются в одну компоненту. С помощью переливаний обновим для вершин корни их компонент и дерево отрезков для hld. Получим n объединений, каждое найдём за $O(log^2(n))$ с помощью hld. Также будет $O(n \cdot log(n))$ обновлений в дереве отрезков из-за переливаний. На каждый запрос будет $O(log^2(n))$ от hld. Итоговая асимптотика $O((n + q) \cdot log^2(n))$.

Теперь приведём красивое решение данной задачи. Начнём с бамбука.

Заменим равенство цен на паре путей на две пары путей с длинами равными максимальной степени двойки, меньшей длины исходного пути (как в sparse table). Теперь длины путей всех огра-

ничений стали равны степеням двойки. Будем перебирать степени двойки в порядке убывания 2^k, для каждого пути длины 2^k заведём вершину в графе, также заведём вершину для каждого такого пути в обратном порядке. Теперь ограничения задают рёбра в таком графе. Проведём их, выделим остовное дерево. Для каждого ребра из остова разделим ограничения на 2 ограничения с длинами путей в два раза меньше и продолжим процесс. На слое с длинами 1 получим нужное нам остовное дерево, которое будет отвечать за первые моменты, когда некоторые пары вершин объединялись в компоненты. Заметим, что с каждого слоя добавится вниз не более $2n$ рёбер, а также от запросов не более $2q$ рёбер. То есть каждый слой отработает за $O((n+q)\cdot\alpha(n))$, где $\alpha(n)$ - среднее время работы в снм, обратная функция Аккермана. Получили решение за $O((n+q)\cdot\alpha(n)\cdot log(n))$.

Для полного решения на дереве для начала разобьём пару путей на три пары соответствующих вертикальных путей(возьмём из 4 конечных вершин этих путей самую близкую к lca пары вершин на её пути, тогда в пару к этому пути поставим вертикальный путь(часть другого пути), теперь получим один вертикальный путь и произвольный путь в дереве, разобьём второй путь по lca и первый по соответствующим длинам). Далее поступим аналогично бамбуку, только место вершины, отвечающей за отрезок, заведём вершину, отвечающую за бинарный подъём в дереве на высоту равную степени двойки. Итоговая асимптотика $O((n+q)\cdot\alpha(n)\cdot log(n))$.

Задача 1C. Королевская задача

Решим задачу для цикла:

- Нужно, чтобы выполнялось два условия: XOR всего цикла равен 0 и XOR на пути между двумя вершинами равен 0.

Решим задачу для ориентированного графа без циклов, где $w_i \leqslant 1$:

- Заметим, что ответ всегда существует, потому что количество путей конечно.
- Посчитаем динамику: $dp[v]$ — XOR всех путей до вершины v из вершины a, а также $p[v]$ — чётность количества путей до вершины v из вершины a. Ответом на задачу будет $dp[b]$.
- Чтобы пересчитать ответ для вершины v, рассмотрим все рёбра (u, w), которые ведут в неё. После этого нужно сделать $dp[v] \oplus = (dp[u] \oplus (w * p[u]))$.

Решим задачу для графа:

- Формулы для предыдущего решения работают в общем случае, поэтому можно просто написать их.
- Можно также отдельно решать задачу для каждого бита.

Основная идея:

- Пусть есть какой-то путь, который проходит через цикл. Тогда XOR на этом пути и цикле должен быть равен 0, чтобы ответ существовал. Поэтому суммарный вклад в ответ всех путей с циклами равен 0.
- Чтобы посчитать ответ, нужно посчитать динамику на DAG-е как во второй и третьей подгруппе на вершинах, через которые не проходит ни один путь с циклом.
- Все циклы лежат внутри компонент сильной связности, поэтому сделаем конденсацию графа.
- Чтобы все циклы внутри компоненты сильной связности имели XOR равный 0, до каждой вершины должен быть единственный возможный XOR на пути до неё.

Используем идею для $n, m \leqslant 1000, w_i \leqslant 1023$:

- Напишем $dp[i][j]$ — можно ли дойти до вершины i с XOR-м j.

- Можно с помощью аналогичной динамики посчитать и суммарный XOR на пути до каждой вершины, через которую не проходит ни один путь с циклом.

Решение для $w_i \leqslant 1$:

- Заметим, что XOR на пути может быть только $0/1$, поэтому мы можем написать аналогичную динамику.

Полное решение (первый способ):

- Независимо решим задачу по каждому биту.
- Получим решение за $O((n+m)\log w_i)$.

Полное решение (второй способ):

- Заметим, что можно посчитать какой-то один XOR на пути до каждой вершины. После этого, чтобы проверить, что до вершины есть единственный XOR на пути до неё, нужно проверить, что вес ребра равен XOR XOR-в путей до двух его концов, которые мы нашли до этого.
- Получим решение за $O(n+m)$.

Задача 1D. Музыкальный фестиваль

Введём для альбома понятие сжатого альбома , который получается из исходного путём удаления всех элементов, кроме тех, которые являются первыми максимумами на соответствующих им префиксах.

К примеру:

Для альбома $[\mathbf{1}, \mathbf{4}, 4, 3, \mathbf{6}, 5, 6]$ сжатым будет альбом $[1, 4, 6]$.

Теперь заметим, что решение исходной задачи сводится к решению этой же задачи, но на сжатых альбомах. Действительно, ответ на них не будет отличаться, ведь если какой-то элемент увеличивал впечатление на обычных альбомах, то он будет увеличивать, если сжать альбомы и наоборот. Далее будет считаться, что предварительно все альбомы были сжаты. Дополнительно введём следующие обозначения $K = \sum k_i, C = \max a_{i,j}$

1 подгруппа Достаточно просто перебрать порядок в котором Маша будет слушать альбомы, и для каждого порядка посчитать количество впечатления, которое она получит, и взять максимум из этих величин. Решение за $O(n!K)$

2 подгруппа Заметим, что ответ равен 2, если есть сжатый альбом $[1, 2]$ или есть сжатые альбомы [1] и [2], в ином случае ответ 1. Решение за $O(K)$

5 подгруппы Отсортируем сжатые альбомы в порядке увеличения последнего числа. Заметим, что если Маша послушала альбом с номером i, то все альбомы с номерами меньше дадут 0 единиц впечатления. Теперь можно посчитать следующую динамику. dp_i — количество впечатления, при условии, что Маша послушала последним i-й альбом. При пересчёте dp_i через dp_j надо посчитать количество, элементов в i-м альбоме, что они больше, чем наибольший в j-м. Решение за $O(n^2 \log K)$, если использовать бинарный поиск.

3 подгруппа Существует не более чем 2^C различных вариантов сжатых альбомов. Тогда удалив одинаковые сжатые альбомы и, применив решение для 5-й подзадачи, можно получить решение за $O(4^C \log K)$.

4 подгруппа Введём dp_c — максимум впечатления, которое можно получить, если бы не было альбомов таких, что в них есть элементы большие, чем c. Тогда, dp_c равно либо dp_{c-1}, либо можно добавить ещё элемент или два, если c является наибольшим элементом для какого-то альбом. Тогда для всех сжатых альбомов, можно пересчитаться через значение dp в точке перед первым элементом альбома, либо через $c-1$. Таким образом для пересчёта достаточно знать для каждого c какие альбомы закончились в этом индексе, а так же для каждого альбома его первый элемент. Решение за $O(K)$

,

Полное решение Обобщим идею предыдущих подгрупп. Для каждого значения c запомним индексы альбомов, которые содержат элемент равный c. Идём в порядке увеличения c, поддерживаем для каждого альбома значение dp_i — максимум впечатления, который можно получить если бы не было элементов больших c и Маша послушала последним i-й альбом. Пусть, для очередного c существует альбом i, что в нём есть песня с крутостью c. Тогда dp_i будет максимумом из dp_i+1 и значения по всем $dp_j + 1$, таких, что наибольший элемент в j-м альбоме меньше чем рассматриваемый элемент i-го, так как она могла послушать этот трек, либо следующим в этом альбоме, либо после того, как послушала какой-то другой альбом целиком. Заметим, что можно хранить максимальный ответ по всем альбомам, для которых наибольшее значение в них меньше c и пересчитываться через это значение. В таком случае, получится решение за $O(K + C)$.

Задача 2A. Задачечка на подстрочечки

Будем обозначать $count(h)$ равным числу подстрок строки h, которые встречаются в наборе s. Соответственно в запросах от нас требуется посчитать $count(h[l_i, r_i])$. Обозначим $pref[i] = count(t[0 : i))$, то есть число подстрок префикса строки t длины i, которые встречаются в наборе S. Обозначим $suf[i] = count(t[i : |t|))$, то есть число подстрок суффикса строки t, начиная с позиции i, которые встречаются в наборе S.

Тогда заметим, что $pref[r] + suf[l] - count(t)$ равно $count(t[l, r])$ минус число подстрок строки h из набора S, которые начинаются раньше позиции l, а заканчиваются позже позиции r. Заметим, что если подстрок второго типа нет, то ответ на запрос легко находится, все нужные значения $pref[l]$ и $suf[r]$ считаются с помощью Ахо-Корасика.

Если строки второго типа есть, то для запроса числа подстрок из S у $t[l, r]$ найдем такую подстроку строки t, которая лежит в наборе S, и её вхождение в t начинается левее l, а заканчивается правее r. Среди всех таких строк найдем ту, у которой правая граница вхождения минимальная. Обозначим эту строку за s_i. Заметим, что строка $t[l, r]$ входит в s_i как подстрока, обозначим это за $s_i[l', r']$. Так же заметим, что в s_i нет подстрок, которые начинаются левее l', заканчиваются правее r' и входят в S как подстрока (иначе для подстроки $t[l, r]$ нашлась бы более близкая справа строка, которая её покрывает. А значит, можно применить исходные рассуждения с префиксами и суффиксами для строки s_i и найти ответ.

Чтобы для запроса $t[l, r]$ найти нужную строку s_i, пройдемся сканирующей прямой по строке t. С помощью Ахо-Корасика для каждой позиции i найдем максимальную подстроку t из набора S, которая заканчивается в позиции i (пусть это строка s_j). Также в структуре куча или set будем хранить начала всех запросов, у которых концы не позже позиции i. Тогда для любого запроса, начинающегося позже начала вхождения s_j, строка s_j будет подходящей. Все такие запросы далее удаляются из кучи.

Таким образом, итоговое решение потребует построения Ахо-Корасика для прямых и развернутых строк, а так же прохода по строке t с поддержкой запросов в куче. Итоговая асимптотика: $O(S + |t| + m \log m)$.

Задача 2B. Покупка подарков

Подзадача 1

Для решения первой подзадачи достаточно реализовать полный перебор действий Саши. В каждом отделе торгового центра у Саши есть два возможных варианта действий, значит всего вариантов его действий порядка $\mathcal{O}(2^n)$. Для каждого варианта необходимо найти стоимость самого дорогого подарка, купленного среди первых магазинов и стоимость самого дорогого подарка, купленного среди вторых магазинов, и вычислить модуль разности их стоимостей. Следует не забыть проверить, что каждая из подруг получит хотя бы один подарок.

Время работы: $\mathcal{O}(2^n \cdot n)$.

,

Подзадача 2

Для решения второй подзадачи зафиксируем номер отдела i, в котором будет куплен самый дорогой подарок для первой подруги, а также номер отдела j ($i \neq j$), в котором будет куплен самый дорогой подарок для второй подруги. Осталось проверить, можно ли купить подарки в остальных отделах таким образом, чтобы подарок a_i был самым дорогим подарком, купленным для первой подруги, а подарок b_j — самым дорогим подарком, купленным для второй подруги.

Научимся выполнять данную проверку за $\mathcal{O}(n)$. Рассмотрим некоторый отдел k ($k \neq i$, $k \neq j$). В данном отделе Саша может либо купить подарок для первой подруги стоимостью a_k, либо купить подарок для второй подруги стоимостью b_k. Если $a_k \leqslant a_i$, то Саша может купить в данном отделе подарок для первой подруги, и подарок a_i все еще будет самым дорогим. Аналогично, если $b_k \leqslant b_j$, то Саша может купить в данном отделе подарок для второй подруги. В противном случае, Саша не сможет купить подарок в данном отделе таким образом, чтобы подарки a_i и b_j были самыми дорогими, и выбранная пара (i, j) не подходит. Осталось среди всех подходящих пар (i, j) выбрать оптимальную.

Время работы: $\mathcal{O}(n^3)$.

Подзадача 3

Для начала отсортируем все отделы по убыванию b_i (а при равенстве — по возрастанию a_i). Теперь переберем отдел i, в котором будет куплен самый дорогой подарок для второй подруги. Заметим, что во всех отделах с номерами $j < i$, Саша должен купить подарок для первой подруги, иначе подарок i не будет обладать максимальной стоимостью среди подарков, купленных для второй подруги. Поэтому сразу найдем значение $m = \max\limits_{j<i} a_j$. Таким образом, мы уже можем получить ответ $|m - b_i|$.

Во всех отделах с номерами $j > i$, для которых $a_j \leqslant m$, Саша может купить подарок для любой из подруг, и это никак не повлияет на ответ. Теперь рассмотрим все отделы с номерами $j > i$, для которых $a_j > m$. Если в некотором из подобных отделов купить подарок для первой подруги, то значение m увеличится, а значит ответ может улучшиться. Поэтому переберем все таким отделы и обновим ответ значением $|a_j - b_i|$.

Время работы: $\mathcal{O}(n^2)$.

Подзадача 4

Отсортируем все отделы по убыванию a_i (а значит, и b_i тоже, так как $a_i = b_i$). Заметим, что в первом отделе Саша купит подарок одной из подруг, стоимость которого максимальна среди всех доступных подарков. Также заметим, что оптимальным решением является покупка подарка второй подруге во втором отделе, потому что любой другой вариант не увеличит стоимость ее подарка, а значит разность стоимостей подарков не уменьшится. Таким образом, ответом является величина $a_1 - a_2$.

Время работы: $\mathcal{O}(n \log n)$.

Подзадача 5

Возьмем решение для подзадачи 3 и оптимизируем его до $\mathcal{O}(n \log n)$. Для начала вместо того, чтобы вычислять величину m на каждой итерации заново, будем поддерживать ее значение в некоторой переменной. Тогда при переходе от отдела $i - 1$ к отделу i будем обновлять значение m следующим образом: $m := \max(m, a_i)$.

Осталось научиться быстро находить оптимальный номер отдела j, такой что $j > i$, $a_j > m$, а также $|a_j - b_i|$ минимально. Выберем на суффиксе массива минимальное a_j, такое что $a_j \geqslant b_i$, а также максимальное a_j, такое что $a_j \leqslant b_i$. Можно заметить, что оптимальным a_j является одно из двух выбранных чисел (нужно также не забыть проверить условие $a_j > m$). Поэтому, достаточно обновить ответ только при помощи них.

Искать данные два элемента можно, используя структуру данных `set`. Будем поддерживать в множестве все a_j, находящиеся на суффиксе. Тогда можно найти нужные два элемента за $\mathcal{O}(\log n)$. При переходе от отдела $i-1$ к отделу i нужно удалить значение a_{i-1} из структуры данных.

Время работы: $\mathcal{O}(n \log n)$.

Задача 2C. Поиск фальшивых монет

Для того, чтобы решить задачу на n запросов, спросим все префиксные суммы. Найдем настоящие элементы последовательности как разности соседних префиксных сумм.

Для того, чтобы решить задачу за $\frac{n}{2} + k$ запросов, спросим все префиксные суммы $n, n-2, n-4, \ldots, n \bmod 2$. Если $pref_i = i + i - 1 + pref_{i-2}$, то числа $i-1, i$ не заменяли. Если это равенство не выполнено спросим $i-1$. Дополнительно мы сделаем $\leqslant k$ запросов. Аналогично сможем узнать все замененные числа.

Далее все ответы на запросы a будем заменять на $\frac{p(p+1)}{2} - a$. Таким образом мы будем узнавать префиксную сумму замененных чисел.

Рассмотрим решение за $k \log n$ запросов. Будем находить замененные числа по одному слева направо. Для того чтобы найти очередное число $\geqslant i$ сделаем бинарный поиск и найдем минимальное j, такое что $pref_j > pref_i$.

Это решение можно реализовать так: рассмотрим функцию `func(l, r,` s_l`,` s_r`)`. Она ищет все замененные числа на отрезке $[l, r]$, если мы знаем что их сумма равна $s = s_r - s_l$.

- Если $s = 0$, то чисел на отрезке нет.
- Если $l \leqslant s \leqslant 2l$, то число на отрезке обязательно одно и оно равно s.
- Иначе спросим $mid = \lfloor \frac{l+r}{2} \rfloor$ и вызовем:
  `func(l, mid,` s_l`,` s_{mid}`)`,
  `func(mid + 1, r,` s_{mid}`,` s_r`)`

Операций все еще $O(k \log n)$, но константа решения становится меньше.

Рассмотрим решение за $2k \log k$ запросов. Мы реализуем `func(l, r,` s_l`,` s_r`)`. Допустим что мы знаем, что чисел на отрезке c штук. Тогда если мы спросим $p = \lfloor \frac{s}{c} \rfloor$, то хотя бы одно замененное число будет слева и хотя бы одно замененное число будет справа. Чтобы найти c сделаем бинарный поиск по нему. Нам не нужно найти точное значение c, главное разделить замененные числа на две части. За $\leqslant \log k$ запросов сможем разделить. Всего будет не более $2k$ вызовов `func`, потому что мы всегда разделяем замененные числа.

В предыдущем решении будем делать бинарный поиск по c в границах $[c_l, c_r]$, где c_l и c_r это $\min/\max$ возможное количество чисел с суммой s из отрезка $[l, r]$. Как только получится разделить замененные числа, вызываем `func` от половинок рекурсивно. Такое решение сделает $\leqslant 2k + \log n$ запросов. Доказательство: можно аккуратно придумать потенциал, который всегда будет уменьшаться на $\geqslant 1$ после каждого запроса.

Для того, чтобы получить полное решение, нужно вставить в прошлое решение все возможные оптимизации и отсечения. Будем реализовывать функцию `func(l, r,` c_l`,` c_r`,` s_l`,` s_r`)`. Изначально `func(1, n, k, k, 0,` s_n`)`. Проверяем что чисел нет, либо одно число, либо весь отрезок, либо весь отрезок без одного числа. Иначе $c_{mid} = \lceil \frac{c_l + c_r}{2} \rceil$, запрос $p = \lfloor \frac{s}{c_{mid}} \rfloor$. Если разделить не смогли, вызываем `func` двигая соответствующие параметры. Если разделить смогли, то находим c_l, c_r для левой и правой половинок. Сначала вызовем `func` от той где $c_r - c_l$ меньше. Когда мы из нее выйдем мы уже будем знать сколько чисел было там и возможно подвинем c_l, c_r второй половинки.

Добавление параметров c_l, c_r в `func` позволяет использовать информацию о том, что замененных чисел было ровно k. Наша функция содержит много `break`-ов и ограничений параметров. За счет этого количество раз, которое мы не сможем разделить множество будет мало. На практике получается $\leqslant k + 30$ операций.

Задача 2D. Путь домой

Подгруппа 1. Если все $w_i = 1$, то ответ это просто длина кратчайшего пути минус p, который можно найти алгоритмом Дейкстры за $O(m \log n)$.

Подгруппа 4. Если все $s_i \leqslant 100$, то можно написать $dp[v][ans] = \max$ money, делая переходы аналогично алгоритму дейкстры. Заметим, что ответ в данном случае не превосходит $S \cdot n$, где $S = \max s_i$. Таким, образом алгоритм работает за $O(S \cdot n \cdot (n+m) \cdot \log n)$

Подгруппа 2. Заметим, что шоу можно делать «отложенно». Как только нам стало не хватать денег, чтобы пройти по ребру, можно сделать несколько шоу заранее среди вершин, которые мы уже прошли, так, чтобы заработать максимальное число денег. Если дан бамбук с концами в 1 и n, то достаточно поддерживать вершину с максимальным w_i на префиксе и делать шоу в ней, если не хватает денег перейти по ребру. Решение в этом случае работает за $O(n)$

Полное решение. Взяв идею с подгруппы 2, можно написать $dp[v][best] = (min\ show, max\ money)$, где v - номер вершины где мы находимся, а $best$ - вершина с макс. w_i, через которую мы уже прошли. Можно показать, что оптимально сначала минимизировать w_i, а потом максимизировать количество денег. Эту динамику можно пересчитывать аналогично решению для подгруппы 4. Асимптотика $\mathcal{O}(mn \log n)$