

## Разбор задачи «Рецепт мармелада»

Заметим, что разница между соседними числами  $b_{i-1}$  и  $b_i$  в массиве  $b$  означает количество чисел  $i$  в исходном массиве. Таким образом, пройдя по всему массиву  $b$ , для каждого из чисел от 1 до  $m$  мы можем восстановить, сколько их было, и, таким образом, восстановить исходный массив (так как известно, что он был отсортирован по неубыванию, ответ единственный).

## Разбор задачи «Стаканчики»

Пусть  $N \leq M$ . Если это не так поменяем значения местами, ответ не изменится. Тогда количество стаканчиков в пирамидке равно  $N \cdot M + (N-1) \cdot (M-1) + \dots + 1 \cdot (M-N+1) = \sum_{i=0}^{N-1} (N-i) \cdot (M-i)$ .

Для прохождения первой группы тестов достаточно было посчитать эту сумму с помощью цикла. Сложность такого решения  $O(t \cdot \min(N, M))$ .

Во второй группе тестов  $N = M$ , тогда  $\sum_{i=0}^{N-1} (N-i) \cdot (M-i) = \sum_{i=0}^{N-1} (N-i)^2$ , что в свою очередь при замене переменной  $j = N - i$  будет равно  $\sum_{j=1}^N j^2 = \frac{N \cdot (N+1) \cdot (2N+1)}{6}$  (общезвестная формула). Сложность  $O(t)$ , так как теперь на каждый запрос отвечаем за  $O(1)$ .

Применим такую же замену в первоначальную формулу, получим  $\sum_{j=1}^N j \cdot (M - N + j) = \sum_{j=1}^N (M - N)j + j^2 = (M - N) \cdot \sum_{j=1}^N j + \sum_{j=1}^N j^2$ . Таким образом мы получили две суммы, которые мы уже умеем считать по формулам. Получаем  $\frac{N \cdot (N+1) \cdot (3M - N + 1)}{6}$ .

По полученной формуле видно, что ответ будет порядка  $N^3$ . Соответственно, чтобы получить полный балл, нужно применить длинную арифметику.

## Разбор задачи «Сеть дорог»

В задаче требовалось построить граф, в котором будет не слишком много вершин, и запустить на нем алгоритм поиска кратчайшего пути. А также, аккуратно разобрать крайние случаи, когда ответа не существует.

В первых группах можно было строить граф, содержащий  $O(k^2)$  вершин. А именно, сделаем вершинами все целые точки с координатами по модулю не превышающими  $k$ . Ребра нужно провести между точками, являющимися соседними в каком-нибудь квадрате, а так же, между концами отрезков радиальных дорог. Несложно показать, что каждая радиальная дорога содержит ровно две целые точки — свои концы. Поэтому, построенный граф является корректным. Затем, запустим алгоритм Дейкстры, и найдем кратчайшее расстояние от начальной точки до конечной. В зависимости от реализации алгоритма, решение может работать за  $O(k^4)$  или за  $O(k^2 \cdot \log k)$ .

Чтобы реализовать решение оптимальнее, нужно заметить несколько фактов. Оставим в рассмотрении только те квадраты, на которых лежит конец отрезка или точка. В каждом квадрате рассмотрим все точки, которые есть во входном файле, и углы квадрата. Расположим их в порядке обхода, и проведем ребра между соседними. Затем, проведем ребра, соответствующие отрезкам. В итоге, получится граф с  $O(n)$  вершин и  $O(n)$  ребер. На нем требуется запустить алгоритм Дейкстры. Итоговая асимптотика  $O(n \cdot \log n)$ .

## Разбор задачи «Равенство»

В этой задаче необходимо было внимательно изучить ограничения для разных подзадач: нет подгруппы, в которую вкладывались бы все остальные. Для каждой подзадачи можно было написать отдельное решение, соединив их вместе. Разберем решения подзадач.

### Подзадача 1. Ручная проверка

В этой подзадаче были даны дополнительные условия  $n \leq 4$ ,  $k = 1$ ,  $t = 1$ . Такие ограничения оставляют небольшой простор для расстановки знаков. Достаточно вручную найти все возможные способы расстановки знаков для всех  $n$ , после чего для каждой из этих расстановок в программе отдельно проверять, не подходит ли она.

### Подзадачи 2 и 3. Перебор

В этих подзадачах достаточно было написать перебор всех расстановок знаков, после чего для каждой расстановки проверить, не подходит ли она. При достаточно аккуратной реализации перебор легко укладывается в ограничение по времени в обеих подзадачах. Реализация для второй подзадачи может быть менее аккуратная, а также проще за счет отсутствия умножения.

### Подзадача 4. Динамика по подотрезкам

Для решения оставшихся подзадач нужно применить динамическое программирование по подотрезкам.

Для краткости записи введем обозначение:  $val[i][j]$  — это число, состоящее из цифр  $a_i, a_{i+1}, \dots, a_j$ , взятое по модулю  $m$ . Иными словами, это остаток от деления на  $m$  числа, которое получается склеиванием всех цифр на отрезке от  $i$  до  $j$ . Все значения  $val$  можно посчитать в программе заранее.

Пусть  $dp[i][j][v]$  — это булево значение, которое истинно, если можно, рассмотрев подотрезок цифр  $a_i a_{i+1} \dots a_j$ , расставить на нем знаки сложения так, чтобы сумма имела остаток  $v$ .

Будем считать значения  $dp$  в порядке увеличения разности  $j - i$ . Чтобы посчитать  $dp[i][j][v]$ , переберем, где будет располагаться последний знак сложения на этом отрезке. Если последний знак сложения будет стоять между символами на позициях  $j'$  и  $j' + 1$ , то мы можем набрать остаток  $v$  на отрезке с  $i$  до  $j'$  в случае, если на отрезке с  $i$  по  $j'$  мы можем набрать остаток  $v - val[j' + 1][j]$ , то есть значение  $dp[i][j'][v - val[j' + 1][j]]$  истинно. Также нужно не забыть про случай, когда на отрезке не будет ни одного знака сложения. В этом случае остаток  $v$  можно получить, только если  $v = val[i][j]$ .

После того, как значения  $dp$  посчитаны, можно решить задачу с помощью динамического программирования по префиксам. Для начала переберем ответ, то есть остаток по модулю  $m$ , который будут давать все выражения. Пусть этот остаток равен  $s$ . Тогда введем еще одну динамику:  $pref[i][j]$  — это булево значение, которое истинно, если первые  $i$  цифр можно разбить знаками равенства на  $j$  выражений, чтобы каждое из них давало остаток  $s$  по модулю  $m$ . Чтобы посчитать  $pref[i][j]$ , переберем, где будет располагаться начало последнего блока. Пусть оно расположено в элементе с номером  $i'$ . Тогда, если значения  $pref[i' - 1][j - 1]$  и  $dp[i'][i][c]$  истинны, значение  $pref[i][j]$  тоже истинно. Если значение  $pref[n][k + 1]$  истинно, то остаток  $s$  подходит. Так как в данной задаче требовалось восстановить ответ, это можно было реализовать либо с помощью стандартной техники — для каждого состояния хранить, из какого состояния мы в него пришли и восстанавливать ответ с конца: сначала найти разбиение на блоки знаками равенства, а потом в каждом блоке найти подходящую расстановку знаков сложения.

Асимптотика времени работы данного решения —  $O(n^3 m)$ .

#### Подзадача 5. bitset

Для решения пятой подзадачи требовалось применить к решению предыдущей подзадачи классическую оптимизацию динамики такого вида: структура данных битовое множество. В языке программирования `C++` эта структура называется `bitset`. Данная структура позволяет проводить массовые операции с набором из  $m$  битов за время  $O(\frac{m}{w})$ , где  $w$  — размер машинного слова. В зависимости от системы  $w$  обычно равно 32 или 64.

Применим `bitset` при подсчете динамики  $dp$ . Заметим, что когда мы пересчитываем значения  $dp[i][j][v]$  для всех  $v$  через значения  $dp[i][j']$ , мы берем все значения из  $dp[i][j']$ , сдвигаем их на одну и ту же величину  $val[j' + 1][j]$  и применяем побитовое ИЛИ ко всем значениям  $dp[i][j]$ . Применим для этого `bitset`. Для этого нужно лишь научиться циклически сдвигать `bitset` на определенное значение  $val[j' + 1][j]$ . Сама структура поддерживает операцию сдвига, однако при обычном сдвиге последние  $val[j' + 1][j]$  значений попадут за границы отрезка  $[0; m - 1]$ . Чтобы вернуть их обратно, сдвинем также `bitset` в другую сторону на величину  $m - val[j' + 1][j]$ , а лишние значения обнулим.

Восстановление ответа в этой подзадаче уже нельзя реализовать с помощью хранения предыдущих состояний динамики из-за наличия массовых операций. Однако, при восстановлении ответа можно перебирать, какое состояние было предыдущим для текущего, и проверять, что из него есть переход в текущее, а также что значение динамики для него истинно. Так можно реализовать восстановление ответа без помощи дополнительной информации.

При наличии навыка работы со структурой `bitset` реализация этого решения будет достаточно проста. Асимптотика времени работы решения —  $O(\frac{n^3 m}{w})$ .

#### Подзадачи 5 и 6. Две динамики по подотрезкам

Для решения последних двух подзадач требовалось реализовать решение, схожее с решением подзадач 4 и 5, с небольшими дополнениями.

Для начала подсчитаем динамику  $dp'[i][j][v]$  — можно ли расставить знаки **умножения** на отрезке  $a_i a_{i+1} \dots a_j$  так, чтобы произведение давало остаток  $v$ . Подсчет динамики  $dp'$  схож с подсчетом

динамики  $dp$  в подзадаче 4, с одним отличием: динамику  $dp'$  проще пересчитывать вперед, так как в противном случае нужно делить по модулю  $m$ , что в общем случае реализуется трудно. Иными словами, надо перебирать, в какие состояния динамики можно перейти из состояния  $dp'[i][j][v]$ : это все состояния вида  $dp'[i][j'][v \cdot val[j][j']]$ .

После того, как динамика  $dp'$  посчитана, можно посчитать динамику  $dp[i][j][v]$ , полностью аналогичную той, которую мы считали в решении подзадачи 4. Разница в подсчете заключается в том, что когда мы перебрали, где будет стоять последний знак сложения на отрезке от  $i$  до  $j$  (пусть он располагается между элементами с номерами  $j'$  и  $j' + 1$ ), у нас все еще остается некоторый простор для выбора: мы можем по-разному расставить знаки умножения на отрезке от  $j' + 1$  до  $j$ . Это значит, что нам дополнительно придется перебрать, какой остаток будет давать произведение на отрезке от  $j' + 1$  до  $j$ . Итак, чтобы посчитать  $dp[i][j][v]$ , переберем, после какого элемента  $j'$  будет стоять последний знак сложения и какой остаток  $v'$  будет давать произведение на отрезке от  $j' + 1$  до  $j$ . Тогда если значения  $dp[i][j'][v - v']$  и  $dp'[j' + 1][j][v']$  истинны, значение  $dp[i][j][v]$  тоже истинно.

Асимптотика времени работы этого решения —  $O(n^3m^2)$ .

Для решения последней подзадачи требовалось снова применить структуру данных `bitset`, полностью аналогично подзадаче 5. Асимптотика времени работы такого решения —  $O(\frac{n^3m^2}{w})$ .