Задача A. Простой шаблон

Есть два случая в данной задаче: когда нет «*» и есть хотя бы одна. В первом случае нужно проверить требуемое условие. Это можно сделать за $O(\sqrt{N})$ времени.

Иначе сделаем число таким, чтобы оно делилось на 3. Для этого достаточно, чтобы сумма цифр делилась на 3. Для этого достаточно заполнять звёздочки последовательно следующей разницей: $9 - \{$текущий остаток от делений суммы цифр на 3$\}$. В таком случае не образуются ведущие нули и работает крайний случай, когда шаблон представляет из себя «*».

Задача B. Возвращение домой

Будем считать, что города — это вершины, а пути между ними – это ребра. Тогда поймем, какие условия обязательны для того, чтобы существовала последовательность действий, оставляющая в итоге ровно 1 ребро.

Первое замечание: все вершины, из которых выходят ребра, всегда должны быть связаны. Это необходимо, потому что за одну операцию у нас добавляется только ребро между вершинами одной компоненты, таким образом в итоге у нас будет количество ребер как минимум равное количеству компонент, а нам нужно в итоге ровно 1 ребро.

Второе замечание: вершин, из которых выходит нечетное число ребер, должно быть не больше 2. Поймем, почему это нужно: за одну операцию у нас число путей меняется только в промежуточной вершине, при этом число ребер уменьшается ровно на 2. Таким образом из всех вершин, у которых изначально было нечетное число ребер у нас будет выходить как минимум 1 ребро, и если таких вершин будет больше 2, то мы в итоге получим больше 1 ребра.

Если какое-то из условий выше не выполнено, то ответа нет. Теперь поймем, что если они выполнены, то мы можем построить ответ. Для этого найдем Эйлеров путь в графе (путь, содержащий каждое ребро ровно 1 раз), потому что для его построения также достаточно этих условий. Тогда используя последовательно используя вершины этого пути $v_1, v_2, \ldots, v_m$, мы пойдем по i от 2 до $m - -1$ и будем превращать ребра (v_1, v_i) и (v_i, v_{i+1}) в ребро (v_1, v_{i+1}).

Задача C. Странная сумма

Давайте разберемся, как считать сумму из условия немного по-другому. Для этого мы можем использовать динамическое программирование и вычислить $dp[i] = dp[i-1] + i \cdot (i+1)/2$, $dp[i]$ - это количество позиций, которые будут сравниваться на равенство для подмассива длины i. Теперь мы вычислим другую сумму, где вместо расстояния Хэмминга мы проверим, сколько позиций в двух подмассивах равны. Тогда исходная сумма на подмассиве длины len будет равна $dp[len]-$ новая сумма. Сейчас мы работаем только с новой суммой. Чтобы найти эту сумму, рассмотрим пары позиций, в которых символы равны, для этого давайте зафиксируем индексы i и j так, чтобы $a_i = a_j, i \leqslant j$. Затем поймем, что эта пара добавляет к сумме число, равное $len - j$. Чтобы понять это, для простоты рассмотрим массив с индексами от 0 до len, тогда пара индексов i и j добавит 1 только для подмассивов $([0; i], [j-i; j]), ([0; i+1], [j-i; j+1]), \ldots, ([0; len-(j-i)], [j-i; len])$. Таким образом, чтобы решить задачу, достаточно вычислить сумму $len - j$ для всех пар.

Теперь давайте покажем, как рассчитать такую сумму с помощью алгоритма МО. Не забудем сжать числа, чтобы мы получили не более n разных цифр и могли использовать массив вместо хэш-таблицы, чтобы хранить статистику для пересчета ответа. Давайте вычислим $lastEq[i]$ - ближайшую позицию перед i, в которой есть цифра a_i. Всю следующую статистику мы считаем для подмассива на текущей итерации алгоритма МО. Мы будем поддерживать следующие значения: $lastNumber[i]$ - индекс последнего элемента на отрезке, равный i, $cnt[i]$ - количество цифр, равное i на отрезке, $lens[i]$ - сумма расстояний между каждой из цифр i до последней цифры i, переменная $cntPair$ - количество упорядоченных пар позиций, в которых цифры равны и переменная ans - текущая сумма для подмассива.

Затем, когда мы переместим правую границу вправо, мы обновим используемые переменные следующим образом: $lastNumber[a_r] = r$, $lens[a_r] \mathrel{+}= cnt[a_r] * (r - lastEq[r])$, $cnt[a_r] \mathrel{+}= 1$, $cntPair \mathrel{+}= cnt[a_r]$, и после этого $cntPair$ будет добавлен в ans. (для каждой пары мы увеличиваем количество доступных подмассивов на 1). Аналогично, при перемещении правой границы влево мы выполняем те же операции в обратном порядке и вместо использования $+=$ используем $-=$, а также $lastNumber[a_r] = lastEq[r]$, если это значение больше левой границы.

При перемещении левой границы влево значения переменных будут обновлены следующим образом: $lens[a_l] += lastNumber[a_l] - l$, $cnt[a_l] += 1$, $cntPair += cnt[a_l]$ и после этого в ответ будет добавлен $lens[a_l] + cnt[a_l] * (r - lastNumber[a_l])$ (добавление суммы для каждой пары, в которой есть индекс l). Аналогично, при перемещении левой границы вправо мы также отменяем операции и используем $-=$ вместо $+=$.

Задача D. Задача для третьеклассника

Основная идея заключается в том, что n может быть представлено в виде $\log n$ произведений и $2 \log n$ сложений. Это можно получить из двоичного представления. Таким образом ответ на запрос меньше $\log n$ или же $a_i \leqslant 2 \log n$. Еще можно показать, что ответ всегда меньше $\log n$, но для решения задачи этого не требовалось.

Разберем случай, когда число произведений меньше $\log n$. В другом случае все аналогично. Обозначим за $dp[i][j]$ минимальное число сложений необходимое для получения числа i, если можно использовать не более j произведений. Есть два вида переходов: последняя операция умножение и последняя операция сложение.

В первом случае переберем все возможные пары состояний динамик, что произведение первых индексов не превосходит n. Это займе $O(n \log^3 n)$ времени.

Во втором случае идея заключается в том, что всегда существует пример, когда одно из слагаемых не превосходит 4. Итоговая асимптотика $O(n \log^2 n)$.

И наконец, каждый запрос требует $O(\log n)$ времени: нужно перебрать все оптимальные пары значений (число сложений, число умножений) для текущего запроса.