# Задача A. Realize

В подгруппе на 40 баллов сумму можно посчитать напрямую за $O(n)$.

Для решения задачи на 100 баллов, воспользуемся следующим фактом: $\lfloor \frac{n}{i} \rfloor$ принимает лишь $O(n^{0.5})$ различных значений.

Доказательство: если $i > n^{0.5}$, то $\lfloor \frac{n}{i} \rfloor < n^{0.5}$, значит принимает не больше $n^{0.5}$ различных значений. Если же $i < n^{0.5}$, то $i$ принимает $O(n^{0.5})$ различных значений, значит и $\lfloor \frac{n}{i} \rfloor$ так же принимает $O(n^{0.5})$ различных значений. Таким образом, мы показали, что у левого множителя в исходной сумме всего корень различных значений.

Давайте найдём все отрезки $i$, для которых $\lfloor \frac{n}{i} \rfloor$ принимает одинаковые значения. Начнём с $i = 1$, и будем постепенно увеличивать $i$, пока оно меньше, чем $n$. Пусть $val = \lfloor \frac{n}{i} \rfloor$, тогда максимальное $i$ с таким же значением выражения будет $i_1 = \lfloor \frac{n}{val} \rfloor$. Часть суммы по этим значения $i$ можно посчитать как $val \cdot (\frac{i_1 \cdot (i_1+1)}{2} - \frac{i \cdot (i-1)}{2})$, а затем пересчитать $i$ как $i_1 + 1$.

# Задача B. Строковедение

В первой подзадаче достаточно перебрать единственную возможную замену за $O(N \cdot 26)$ и пересчитать ответ.

Заметим, что все, что нас интересует про строку, для того, чтобы посчитать величину ответа — массив длины 26 с количеством вхождений каждой буквы. Также отметим, что если какие-то буквы не будут встречаться в строке, то выгоднее сделать так, чтобы это были буквы, которые изначально были самые редкие.

Тогда можно отсортировать все 26 букв (вне зависимости от того, встречаются они в строке или нет) по количеству вхождений, и перебрать за $O(26)$ сколько первых букв не будут встречаться в итоговой строке в порядке этой сортировки. Заметим, что некоторые из букв, которых ранее не было в строке, иногда выгодно в нее добавить, уменьшая количество других букв. Например, в строке *aabbccd* выгодно каждое второе вхождение букв заменить на новую, добившись нулевой неравномерности: *aebfcgd*.

Теперь осталось определить, какого максимального минимума и какого минимального максимума по количеству вхождений букв можно добиться. Если у нас осталось $z$ различных букв, то $min \leqslant \lfloor \frac{N}{z} \rfloor$ и $max \geqslant \lceil \frac{N}{z} \rceil$. Тогда, можно определить достижимые значения минимума и достижимые значения максимума независимо. Для проверки достижимости достаточно за $O(26)$ пройтись по массиву количеств, посмотреть какие элементы вылезают за границу, просуммировать разницы, и, если сумма не больше, чем $k - r$, то такая граница достижима. Здесь $r$ — количество операций, которое мы потратили на удаление редких букв, они заранее внесли вклад в увеличение минимума, но не в уменьшение максимума. Из всех достижимых минимумов и максимумов мы выберем пару с минимальной разницей.

Когда мы определили, какие буквы мы удалим, какой будет минимум и максимум в оптимальном ответе, его нужно восстановить. Это можно делать жадно: заменяем лишние вхождения самых частых букв теми буквами, у которых вхождений слишком мало (или нет совсем), или же, когда таких не осталось, по очереди увеличиваем количество самых редких букв.

Таким образом, мы получаем решение за $O(26^2 \cdot N)$, которое заходило на 60 баллов.

Для получения 100 баллов достаточно заметить, что вместо перебора верхних/нижних границ, можно было сделать два бинарных поиска (так как обе функции выпуклые), получив асимптотику $O(N \cdot 26 + 26^2 log N)$.

# Задача C. Мощный процессор

Научимся решать задачу за $O(8 \cdot n)$ на запрос поиска НВП на отрезке $[l; r]$. Для этого достаточно поддерживать динамику $dp[i][j]$ — пусть мы находимся в элементе $i$, последний элемент $\leqslant j$, а значение — максимальная длина такой возрастающей последовательности. Тогда в позиции $i$ мы либо берём элемент, и переходим в $dp[i+1][k]$, либо не берём элемент на позиции $i$ и переходим в $dp[i+1][j]$. Такое решение набирало 22 балла.

Если $a[i] \leqslant 1$ в любой момент времени, то на запрос seq надо проверить, что самый левый ноль стоит левее самой правой единицы. Чтобы это проверить, храним в дереве отрезков самый левый

ноль, самую левую единицу, самый правый ноль и самую правую единицу. При запросе xor у нас либо $x = 0$ (тогда ничего не делаем), либо $x = 1$ (тогда меняем местами 0 и 1 на отрезке).

Решение работает за $O((n + q)logn)$.

В подгруппе 6 давайте будем считать уже другую динамику: $dp[k][i]$ — минимальная позиция в массиве, такая что мы набрали возрастающую последовательность длины $k$ и закончили на элементе со значением не больше $i$. Тогда необходимо сделать переходы двух видов: в $dp[k][i+1]$ с таким же значением, или же в $dp[k+1][i+1]$, со значением, равным первому элементу в массиве после позиции $dp[k][i]$ с величиной равной $i$. Тогда, если поддерживать дерево отрезков, где в вершине мы будем хранить про каждое значение до 8 минимальный индекс на отрезке массива, где оно встретилось, то с помощью запроса минимума на отрезке мы сможем сделать переход второго типа. Решение работает за $O(q \cdot 8^2 \cdot \log n + 8 \cdot n)$.

На полный балл достаточно поддерживать такое дерево отрезков и реализовать там массовые операции. Действительно, во время применения $xor$ мы можем пересчитать новые значения минимальных позиций каждого числа от 0 до 7 за $O(8 \cdot \log n)$. Тогда решение будет работать за $O(q \cdot 8^2 \cdot \log n)$ и набирать 100 баллов.

Альтернативным решением будет перебрать все различные возрастающие подпоследовательности — их всего $2^8$, и делать запросы про минимальные позиции с нужными значениями на отрезке к тому же дереву отрезков. При аккуратной реализации оно также заходит на 100 баллов.

# Задача D. Long shot

В подгруппе на 5 баллов можно было сдать полный перебор с отсечениями. Поддерживаем, к каким комнатам у нас сейчас есть доступ, пытаемся применить какой-то контракт.

Рассмотрим упрощенную версию задачи, где нужно получить доступ только к одной комнате. Рассмотрим минимальную по стоимости цепочку контрактов, которая позволит это сделать. Заметим, что тогда каждый контракт открывает доступ к комнате со следующим контрактом, ведь иначе можно было бы сократить путь из контрактов. Тогда, достаточно хранить только какой контракт мы взяли последним, и можно построить граф, где вершинами будут контракты, а ребрами — стоимости их активации. Начать мы можем из любого контракта, который доступен из первоначально открытой комнаты. Тогда расстояния от контрактов до терминального состояния (контракта дающего доступ в 1 комнату) можно предподсчитать за $O(k^2)$, а затем уже находить ответ для каждой комнаты суммарно за линию.

Если же нам нужно получить доступ в две комнаты, то по сути нам нужно найти два пути по контрактам. Нам нужно учесть, что пути могут иметь общие контракты, за которые не нужно дважды платить маной. Заметим, что если они пересекаются, то имеют только общий префикс, иначе можно было улучшить ответ, удалив лишние ребра/вершины. Тогда можно отдельно посчитать расстояния до каждой из ключевых комнат (1 и $n$), а затем обновить начальное расстояние для вершины как суммы этих двух, и запустить алгоритм Дейкстры ещё раз. Таким образом, получаем решение за $O(k^2 + n)$, которое проходит все группы, кроме последней.

В последней группе нужно было сооптимизировать алгоритм Дейкстры. Один из способов сделать это — сжать граф с помощью дерева отрезков. Из комнаты с контрактом нам нужно провести ребро весом стоимость контракта во всем контракты, отрезок комнат которых его содержат. Для этого добавим $O(n)$ новых вершин, в форме дерева отрезков. Проведем ориентированные ребра веса ноль, направленные от сыновей к предку, и заранее проведём $O(logn)$ ребер для каждого контракта, из вершин, отвечающих за отрезок комнат, которые он открывает, в его вершину. Тогда достаточно ещё провести ребро из контракта в вершину дерева отрезков на нижнем уровне, отвечающую за комнату, в котором он лежит, и с весом в его стоимость, чтобы находить кратчайшие пути на этом графе. Таким образом, Дейкстра будет работать за $O((n + k)log^2 n)$, и при аккуратной реализации такое решение заходит на 100 баллов.