Разбор задачи «Homework»

Обозначим число символов в строке A как n. Если какой-то символ встречается больше, чем $\frac{n}{2}$ раз, то ответ «`IMPOSSIBLE`», потому что в этом случае множество позиций, на которых этот символ встречается в первой строке, и множетсво позиций, на которых этот символ встречается во второй строке, обязательно пересекутся.

Иначе, ответ всегда существует. Можно придумать много разных конструкций, приведем самую простую известную нам. Строку B можно выбрать любую, но удобнее всего взять строку A и отсортировать в ней символы: это означает, что позиции каждого символа образуют подотрезок. После этого, возьмем строку C равной строке B, но циклически сдвинутой на $\frac{n}{2}$. Отрезок для каждого символа также сдвинется на $\frac{n}{2}$, но, так как число вхождений каждого символа не больше $\frac{n}{2}$, то эти отрезки в B и C не будут пересекаться.

Ниже приведена реализация решения на Python:

```
A = input()
n = len(A)
B = sorted(A) # Отсортируем A
C = B[n//2:] + B[:n//2] # Получим C как циклический сдвиг B на n // 2
if any(B[i] == C[i] for i in range(n)): # Проверим, получился ли верный ответ
    print("IMPOSSIBLE") # Если нет, то IMPOSSIBLE
else:
    print("".join(B)) # Если да, то выводим B и C
    print("".join(C))
```

Разбор задачи «Matryoshka Inc»

Сформулируем задачу формальнее: дан массив записей чисел (возможно, с ведущими нулями), у каждого числа можно переставить цифры произвольным образом. Необходимо максимизировать длину наибольшей возрастающей последовательности.

Нахождение НВП — классическая задача, решающаяся динамическим программированием, однако, здесь мы не можем хранить лишь `dp[i]`, поскольку нам интересно также и то, чему равен последний элемент. Если включить его в состояние, получится решение, проходящее некоторые частичные подгруппы.

Рассмотрим чуть другое решение с помощью ДП: будем итерироваться по числам слева направо (пусть текущий индекс — i) и поддерживать `dp[j]` — минимальное число, на которое заканчивается возрастающая последовательность длины j. Этой информации будет достаточно, чтобы переходить к следующему числу.

Как обработать очередное число из массива? Переберем, к возрастающей последовательности какой длины мы припишем число на позиции $i+1$, пусть эта длина j. Нужно теперь переставить цифры в $i+1$-м числе так, чтобы получилось число большее, чем `dp[j]`, а из таких — минимально. Это также весьма стандартная задача: постараемся взять число, с наибольшим общем префиксом, а на первой несовпадающей позиции поставим минимальную цифру из тех, что больше. Вызовем эту подзадачу для каждой длины j, получим ответ.

Итоговая сложность решения выходит $O(n^2B)$, где B — десятичная длина чисел во входе.

Разбор задачи «Password Lock»

Давайте вместо чисел на ячейках будем смотреть на их остатки при делении на k. Числа с одинаковым остатком мы почти всегда можем ставить рядом, кроме случаев остатка 0 и $\frac{k}{2}$ (если k четное).

Давайте сначала попробуем поставить все ячейки в порядке возрастания остатка. Тогда у нас может быть три места, которые не подходят под условие:

1. два остатка 0 рядом;
2. два остатка $\frac{k}{2}$ рядом (такое может быть только в случае четного k);

3. два остатка x и $k - x$ рядом, $x \neq k - x$ (такое может быть максимум в одном месте).

Давайте временно уберем все ячейки с остатком 0 и $\frac{k}{2}$. Чтобы избавиться от третьего случая, нужно просто вставить любой другой остаток между этими элементами. Можно взять либо 0 (если он есть), либо $\frac{k}{2}$ (если он есть), либо ячейку из начала (если там другой остаток), либо ячейку из конца (если там другой остаток). Берем именно из начала или конца, чтобы не испортить ничего и не добавлять третий случай в других местах. Понятно, что если у нас всего два различных остатка, получить подходящую последовательность не получится.

Теперь попробуем обратно вставить ячейки с остатками 0 и $\frac{k}{2}$. Пусть нулей меньше (в случае меньшинства $\frac{k}{2}$ действуем аналогично). Тогда вставляем их всех в начало. После вставляем все $\frac{k}{2}$ через одного. Поскольку их больше, то соседних нулей не будет. Если $\frac{k}{2}$ слишком много, и мы не смогли вставить все, то это означает, что их больше половины от всех ячеек; тогда получить подходящую последовательность не получится.

Таким образом вы избавились от всех трех возможных случаев.

Разбор задачи «The Name of the Fourth Problem»

В математике такая последовательность известна как *Golomb sequence*. Чтобы решить первые несколько подзадач, нужно разобраться в том, как она работает (либо найти явную формулу вида $a(n) = 1 + a(n - a(a(n)))$ и запрограммировать вычисление частичных сумм.

Чтобы уметь решать задачу для больших n, необходимо использовать специальную структуру последовательности. Заметим, что в ней достаточно много повторов, которые, к тому же, идут в последовательности подряд. Используем `Кодирование длин серий (Run-length encoding)`. Вместо того, чтобы хранить числа явно, будем сжимать отрезки подряд идущих в пару (число, количество). Считать префиксные суммы на такой структуре сложнее, но все еще возможно: считаем префиксные суммы по количествам чисел (чтобы находить границу запроса, например, двоичным поиском), а также по сумме чисел в отрезке (иначе говоря, по произведению значения на количество). Такое решение работает при $r_i \leqslant 10^{10}$ и чуть дальше.

Чтобы получить 100 баллов за задачу, нужно проделать еще один аналогичный шаг. Рассмотрим отрезки, полученные в предыдущем решении. По аналогичным рассуждениям, в этих отрезках параметр «количество» будет монотонно не убывать, и будет много повторяющихся значений (на самом деле, размеры этих отрезков в точности соответствуют изначальной последовательности, поскольку она само-описывающая). Проделаем сжатие снова, получится, что мы храним информацию вида «сейчас идет отрезок, содержащий числа от a до b, каждое повторяется k раз». Аналогично, на этих отрезках можно построить префиксные суммы и находить двоичным поиском границы запроса. Решение работает за предподсчет (необходимо сохранить $L < 10^6$ отрезков) + $O(\log L)$ на запрос.

Разбор задачи «Comparing Theories»

Посмотрим, как можно подступиться к подсчету таких троек. Во-первых, будем считать не отличающиеся тройки, а совпадающие, а в конце вычтем их количество из $\binom{n}{3}$.

За $O(n^4)$ или $O(n^3)$ можно написать следующее решение: переберем все тройки номеров листьев, вычислим вершину, в которой «сходятся» три пути. Она будет являться родителем двух из трех листьев, проверим, совпадает ли эта пара в двух деревьях. Вершина, в которой сходятся три пути является та, которая в списке $(\mathrm{LCA}(A, B), \mathrm{LCA}(A, C), \mathrm{LCA}(B, C))$ встречается ровно один раз. В зависимости от скорости нахождения LCA такое работает от $O(n^4)$ до $O(n^3)$.

Как считать тройки быстрее? Рассмотрим LCA трех вершин в первом дереве. У этой вершины два поддерева: раскрасим (во втором дереве!) листья из первого поддерева в красный цвет, а из второго — в синий. Вершины снаружи этого поддерева будут бесцветными. Посмотрим на произвольную вершину во втором дереве, представим, что она также является LCA трех листьев, но во втором дереве.

Сколько есть способов выбрать три листа, чтобы структуры их деревьев совпадали, а LCA в первом и втором дереве были равны выбранным вершинам? Поскольку эти вершины (во втором дереве) должны быть в точности LCA трех листьев, два листа должны прийти из одного поддерева, а один —

из другого. Обозначим за R_1, R_2, B_1, B_2 число красных/синих листьев в левом/правом поддереве вершины во втором дереве. Тогда число согласованных троек равно $\binom{R_1}{2}B_2+\binom{R_2}{2}B_1+\binom{B_1}{2}R_2+\binom{B_2}{2}R_1$. Просуммируем эту величину по всем вершинам второго дерева, получим ответ. Такое решение можно реализовать за $O(n^2)$.

Предположим, что у нас есть структура данных для второго дерева, позволяющая делать две операции:

- Изменить цвет листа.
- Просуммировать формулу выше по всем вершинам.

Как реализовать такую структуру данных, обсудим позже. Сначала обсудим, как за $O(n \log n)$ запросов к такой структуре найти ответ. Запустим обход в глубину, который будет раскрашивать листья в красный и синий для каждой вершины. Инварианты: перед заходом в вершину v все листья в ее поддереве — красные. После выхода из вершины все листья в поддереве будут без цвета. Он будет действовать следующим образом:

1. Перекрасим все листья в **меньшем** из двух поддеревьев v в синий, спросим у структуры данных ответ для текущей вершины.
2. Снимем раскраску в меньшем поддереве.
3. Запустим обход для **большего** поддерева, пререквизит выполнен.
4. Перекрасим все листья в **меньшем** из двух поддеревьев v в красный, пререквизит выполняется, запустимся рекурсивно.
5. Оба рекурсивных обхода почистили свое поддерево, можно выходить.

Внутри каждой вершины мы делаем $O(k)$ операций, где k — размер меньшего поддерева. Суммарно по всем вершинам это оценка дает $O(n \log n)$ обращений к структуре данных.

Структуру данных можно реализовать с помощью Heavy-Light decomposition ($\log^2 n$ или $\log n$ на запрос), где необходимо изменять цвет листа и обновлять все количества на пути до корня. Также, так как эта задача действительно важна в вычислительной биологии, есть множество статей, которые считают расстояние троек/четверок. Одна из структур данных, которую называют *Hierarchical Decomposition Tree* позволяет отвечать на этот запрос также за $O(\log n)$ на запрос. Подробнее ее реализацию можно прочитать в статье `https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-S2-S18`.