Задача A. Перерыв на обед

Нужно перебрать заведение, в котором будет обедать Лэнс. После этого, вычислить длины двух отрезков по теореме Пифагора, и обновить ответ суммой длин отрезков плюс t_i.

Задача B. Починка массива

Заметим пару полезных фактов: если с помощью операций перенести k элементов в начало и m элементов в конец массива, то:

- Можно выполнять операции с такой очередностью, что перенесенные элементы будут в любом необходимом порядке, например отсортированном.
- Элементы, которые перенесены в начало являются k наименьшими числами исходного массива, а в свою очередь элементы, перенесенные в конец должны быть m наибольшими числами в массиве.

Таким образом, чтобы массив был отсортирован, все элементы, которые не переносились должны быть в отсортированном порядке относительно друг друга.

Для облегчения задачи выполним масштабирование: отсортируем первоначальные элементы в отдельном массиве, а затем уберем повторы. Затем в исходном массиве заменим элементы, на их индексы в полученном массиве. Например, если изначально массив выглядел, как $\{3, 1, 2, 3, 40, 55\}$, после масштабирования он будет выглядеть следующим образом: $\{2, 0, 1, 2, 3, 4\}$. Заметим, что так как масштабирование не влияет на порядок элементов, ответ на задачу для исходного массива и нового будет идентичным. Более того, теперь в массиве все значения не превосходят его длину.

После масштабирования выполняется следующее свойство, если в массиве есть элемент x, то в нем также есть элемент $x + 1$, либо x является наибольшим элементом массива. Тогда заметим, что после удаления из массива элементов, к которым была применена операция оставшиеся элементы будут образовывать последовательность следующего вида: $\{x, x, \ldots, x, x{+}1, x{+}1 \ldots x{+}1, \ldots x{+}q\}$, то есть разобьется на отрезки из соседних чисел. Иначе массив не будет отсортирован. Тогда заметим:

- Если к хотя бы одному элементу со значением x применяется операция по переносу в начало, то нигде среди оставшихся элементов кроме как в начале не могут находится элементы с таким же значением.
- Аналогичное верно для переносов в конец.

Для начала для каждого значения массива посчитаем его первое и последнее вхождение в исходный массив, составим из них отрезок $[l_x; r_x]$. Тогда оставшиеся элементы представляются, как последовательно соединенные отрезки, а также для каждого отрезка верно:

- Все значения за которые он отвечает были переставлены, тогда отрезок не входит в оставшиеся элементы.
- Все элементы из этого отрезка остались нетронутыми, тогда он входит в оставшиеся элементы.
- Некоторые (не все) элементы со значениями, за которые отвечал отрезок, могли быть перенесены вперед и тогда отрезок $[l_x, m_x]$ будет являться началом последовательности оставшихся элементов, где m_x некоторое вхождение x.
- Аналогично для переносов в конец.

Некоторые отрезки вполне могут «не мешать друг другу», например, если рассмотреть некоторое значение x и будет верно, что $r_x < l_{x+1}$, заменим эти два отрезка на отрезок $[l_x; r_{x+1}]$. После всех возможных замен получится, что последовательность оставшихся элементов выглядит следующим образом: Либо начало отрезка и конец следующего за ним, либо три подряд отрезка, у крайних из которых взяты только части. Все нужные значения m_x необходимые для нахождения частей отрезков предлагается искать двоичным поиском.

Итоговая сложность решения $O(n \log n)$.

Задача C. Кодовый замок

В первой подзадаче нужно было проверить, что все поворотные ручки достижимы из единственного центрального элемента.

Назовем интересными поворотными ручками те, из которых достижим центральный элемент и по вертикали, и по горизонтали. Если центральных элементов не больше 5 и решение существует, количество интересных поворотных ручек не может превышать 20. Поэтому, если их больше, решения точно нет. А если меньше, то можно перебрать 2^{20} вариантов ориентации этих ручек. Ориентация остальных ручек определяется однозначно. После чего нужно только проверить, является ли текущая ориентация решением.

Заметим, что у каждой ручки есть выбор из максимум двух центральных элементов, к которым она может быть в итоге присоединена. Заведем для каждой ручки булеву переменную, обозначающую к какому из двух центральных элементов она будет присоединена. Допустим, ручка будет ориентированна горизонтально. Тогда, все ручки между данной и центральным элементом, к которому она будет присоединена, тоже должны быть ориентированны горизонтально. А если любая из этих ручек ориентированна вертикально, то и данная ручка должна быть ориентированна вертикально (потому что она не может быть ориентированна горизонтально). Аналогичные условия, если ручка будет ориентированна вертикально. Несложно доказать, что эти условия являются необходимыми и достаточными. Заметим, что все эти ограничения могут быть сформулированны в формате 2-SAT. При этом, в нем будет $O(n^2)$ переменных и $O(n^3)$ клозов. Решение 2-SAT может быть найдено за линейное от количества клозов время. Таким образом, мы получили решение за $O(n^3)$ для третьей подзадачи.

Чтобы решить последнюю подзадачу, нужно оптимизировать структуру графа, который получается при решении 2-SAT. Заметим, что мы всегда проводим ребра в префикс вершин, если смотреть от какого-то центрального элемента. Поэтому, для каждого центрального элемента в каждом из четырех направлений для каждого префикса сделаем фиктивную вершину, из которой проведем два ребра: в фиктивную вершину для префикса на одну вершину короче, и в эту вершину. Теперь вместо того, чтобы проводить ребра в отрезок вершин, проведем одно ребро в фиктивную вершину, из которой будет достижим этот же отрезок. Количество вершин осталось $O(n^2)$, и количество ребер тоже стало $O(n^2)$.

Задача D. Побег с горной базы

В первой подзадаче можно перебрать все варианты расстановки вертолетов, и выбрать оптимальный. Решение работает за $O(2^n \cdot n)$.

Заметим, что вертолеты всегда выгодно ставить в листы дерева. И если количество листов меньше, чем k, то ответом является n. Решим вторую подзадачу методом динамического программирования. Будем перебирать листы в порядке, в котором они встречаются в каком-то эйлеровом обходе дерева. Состоянием будет пара чисел (i, v) — количество вертолетов, которые уже поставлены, и номер последнего листа, в который был поставлен вертолет. Допустим, мы хотим поставить вертолет в лист u, тогда нужно обновить значение $dp[i+1, u]$ значением $(dp[i, v] + depth(u) - depth(lca(u, v)))$, где $depth$ — глубина вершины, а lca — наименьший общий предок. Получившееся решение работает за $O(k \cdot n^2)$ или $O(k \cdot n^2 \cdot \log(n))$ в зависимости от реализации нахождения наименьшего общего предка.

Для решения третьей подзадачи, нужно оптимизировать состояние динамического программирования из предыдущего абзаца для графов с маленькой высотой. Вместо того, чтобы хранить номер последнего взятого листа, будем хранить $depth(lca(v, u))$, где v — последний взятый лист, а u — текущий лист. Теперь, пусть мы переходим от листа номер u к следующему листу номер q. Несложно доказать, что $depth(lca(v, q)) = \min(depth(lca(v, u)), depth(lca(u, q)))$. Поэтому, при переходе от листа к следующему, нужно обрезать значение глубины lca с последней взятой вершины до глубины lca текущего и следующего листа. В остальном, переходы аналогичны переходам из предыдущего абзаца. Получилось решение, работающее за $O(k \cdot D \cdot n \cdot \log(n))$, где D — максимальная глубина вершины. Заметим, что оно укладывается в ограничения как третьей, так и второй подзадач.

Заметим, что самую глубокую вершину всегда выгодно взять. Допустим, мы ее не возьмем. Тогда начнем подниматься из нее до корня, пока не встретим вершину, из которой достижима

взятая. Заменим эту взятую вершину на ту, из которой поднимались, и ответ не ухудшится. Поэтому, возьмем самую глубокую вершину в ответ, и удалим из дерева все вершины на пути от взятой вершины до корня. При этом, дерево может распасться на несколько. Теперь, аналогично можно доказать, что снова выгодно взять в ответ самую глубокую вершину (теперь глубина вершины считается от корня того дерева, в котором она лежит). Снова возьмем ее и удалим из дерева путь от нее до корня. Будем делать там k раз. Каждый раз мы находим самую глубокую вершину и удаляем путь за $O(n)$. Поэтому, получилось решение, работающее за $O(n \cdot k)$.

Отсортируем все вершины по глубине. Будем рассматривать их в таком порядке. Когда мы рассматриваем вершину v, будем подниматься из нее до корня, пока не встретим помеченную вершину. Обозначим за c_v количество непомеченных вершин, по которым мы успели пройти. После чего пометим все вершины, по которым прошли. Оказывается, что ответом является сумма k максимальных чисел среди c_v. Нужно доказать, что для вершин, которые войдут в ответ, c_v будет равно глубине в момент удаления в алгоритме из предыдущего абзаца. Доказательство оставим читателю в качестве упражнения :). Получили решение, работающее за $O(n)$.