Задача A. Стать сильнее

Автор задачи и разработчик: Даниил Орешников

Каждое описанное в задаче устройство — стек, то есть чем раньше в него добавляется компонент, тем позже он будет вынут. Таким образом, компоненты, добавленные раньше, будут находиться в устройстве строго дольше, чем те, которые были добавлены позже.

Осталось только заметить, что если $|a_i - a_j| \leqslant 1$, компоненты i и j не могут находиться в одном устройстве. Если между любыми двумя действиями с устройством должно пройти не меньше секунды, то между моментами их добавления прошло не меньше секунды, и между моментами их вынимания прошло не меньше секунды, таким образом соседние в устройстве компоненты должны иметь время работы, отличающееся хотя бы на 2.

Для решения первых двух подзадач достаточно было написать полный перебор распределения компонентов по устройствам. Более того, в первой подзадаче можно было просто рассмотреть несколько возможных случаев, чтобы определить, хватит ли одного, двух или трех устройств.

Ограничения третьей подзадачи гарантируют, что a_i даны в возрастающем порядке. Таким образом, не бывает совпадающих элементов, но могут присутствовать элементы, отличающиеся на 1. Достаточно проверить, существует ли такое i, что $a_{i+1} = a_i + 1$. Если такое есть, понадобится хотя бы два устройства, и двух достаточно — можно распределить все компоненты с четным временем работы в порядке его убывания в первое устройство, а все с нечетным — во второе. Если же $a_{i+1} - a_i > 1$ для всех i, то достаточно одного устройства.

Для решения всех оставшихся подзадач в начале необходимо отсортировать все a_i по возрастанию или по убыванию. В подзадаче номер 4 гарантируется, что все a_i четны, а значит необходимо и достаточно, чтобы компоненты с совпадающим временем работы были в разных устройствах. В отсортированном массиве найти максимальное количество одинаковых элементов можно за линейное время.

Для оставшихся двух подгрупп требовалось заметить, что если какие-то k подряд элементов в отсортированном массиве a_i отличаются друг от друга не более, чем на 1, но при этом $a_{i+k} - a_i > 1$ для всех i, то ровно k устройств будет необходимо достаточно — поместим в первое устройство компоненты с индексами 1, $k+1$, $2k+1$, $\ldots$, во второе — с индексами 2, $k+2$, $2k+2$, $\ldots$, и так далее, в последнем устройстве будут компоненты с индексами k, $2k$, $3k$, $\ldots$.

Можно было найти такое k за время $\mathcal{O}(n^2)$, просто двигаясь вперед от каждого i по отдельности. А можно было и за линейное время, заметив, что чем больше i, тем больше будет первое j, для которого $a_j - a_i > 1$. Это означает, что можно применить метод двух указателей — искать j, соответствующее левому индексу $i+1$, начиная с j, найденного на предыдущем шаге для левого индекса i. Метод двух указателей работает за линейное время, итоговое время работы — $\mathcal{O}(n \log n)$ из-за сортировки.

Задача B. Биомаркеры

Автор задачи и разработчик: Константин Бац

Формально, в задаче требовалось найти максимальное кратное трем число m такое, что последовательность цифр числа m является подпоследовательностью цифр числа n.

Для решения первых двух подзадач, когда в числе n не больше шести значащих цифр, было достаточно просто перебрать все числа от 0 до n, кратные трем, и среди таких выбрать максимальное, у которого цифры являются подпоследовательностью цифр n. Проверку на то, что цифры одного числа являются подпоследовательностью цифр другого, можно осуществить за линейное от длин чисел время. Для этого достаточно перебирать цифры числа и для каждой находить минимальное следующее за предыдущей вхождение этой цифры в число n.

В третьей подзадаче k, количество цифр числа n, было ограничено сверху числом 18. Количество всех возможных подпоследовательностей цифр n равно $2^k < 10^6$, что позволяет перебрать такие подпоследовательности, проверить их на кратность трем и выбрать из них ту, которая дает максимальное число. Построить и сравнить два не более чем k-значных числа можно за время $\mathcal{O}(k)$, и время работы такого решения — $\mathcal{O}(k \cdot 2^k)$.

Для решения последующих подзадач требовалось свойство кратности целых чисел трем: *остаток от деления числа x на три равен остатку от деления суммы цифр числа x на три.*

Давайте посчитаем сумму цифр числа n, назовем ее s, а остаток от деления ее на три — r. Если s кратно трем, то есть $r = 0$, то ответ на задачу — число n. Иначе, рассмотрим случаи $r \neq 0$. Для простоты будем называть *противоположным остатком* к r число $2 - r$, то есть для одного — два, а для двух — один.

Если число n имеет остаток $r \neq 0$, то верно хотя бы одно из двух:

- в числе n есть не меньше одной цифры с остатком r;
- в числе n есть не меньше двух цифр с остатком, противоположным r.

Заметим, что если в числе n нет нулей, то чтобы сделать его кратным трем, потребуется удалить не больше двух цифр. Действительно, воспользуемся утверждением выше: если в числе есть цифра с остатком r, можно удалить только ее, иначе можно удалить две цифры с остатком, противоположным r остатком.

Ограничения четвертой подзадачи позволяли проверить оба случая, явно перебрать позиции удаляемых цифр, и выбрать максимальное число из тех, которые получаются после удаления. Заметим, что чем больше цифр мы удаляем, тем меньше число, поэтому удалять больше двух цифр из числа n не имеет смысла.

Для хранения числа n, промежуточных вычислений и сравнений можно использовать строки или массивы чисел. Сравнение двух чисел будет выполняться за $\mathcal{O}(k)$, а на перебор позиций уйдет $\mathcal{O}(k^2)$. Тогда время работы такого решения равно $\mathcal{O}(k^3)$.

Теперь рассмотрим более внимательно имеющиеся варианты. Пусть число n не содержит нулей, и из него можно удалить одну цифру, чтобы оно стало кратно трем. Тогда посмотрим на все позиции цифр, имеющих остаток r по модулю три.

Среди них могут быть такие $i \leqslant k - 1$, для которых следующая цифра на позиции $i + 1$ больше, чем текущая, то есть $n_i < n_{i+1}$. Докажем, что если удалить цифру с минимальной такой позиции i, то получится максимальное число, кратное трем, которое можно получить, удалив из n одну цифру. Действительно, при удалении цифры на позиции i мы получаем число, имеющее общий префикс с n длины $i - 1$, после которого следует цифра n_{i+1}.

Если обозначить за m число, образуемое префиксом числа n длины $k - 1$, то удаление цифры, большей, чем следующая за ней, приведет нас к получению числа, меньшего m. А удаление цифры на позиции i, меньшей, чем следующая за ней, позволит получить число, большее m уже в i-й цифре. Чем меньше будет такое i, тем больше будет конечное число.

Если же в записи числа n нет такой позиции $i \leqslant k - 1$, что остаток цифры на этой позиции равен r, и $n_i > n_{i+1}$, то выгодно удалить цифру, имеющую остаток r, находящуюся на максимальной из

всех таких цифр позиции. Это можно доказать аналогичными приведенным выше рассуждениями.

Если в числе n нет цифр с остатком r, то придется удалить две цифры противоположного остатка. С помощью доказанных выше утверждений можно свести задачу удаления двух таких цифр к удалению сначала одной такой цифры (либо находящейся как можно левее и меньшей своего правого соседа, либо находящейся как можно правее) по очереди.

Пятая подзадача выделялась тем, что цифры шли в порядке неубывания, что, с одной стороны, гарантировало отсутствие нулей, а с другой стороны, позволяло проще искать индексы интересных нам цифр. Действительно, если для любого i верно, что $n_i \leqslant n_{i+1}$, то достаточно взять минимальный индекс i с остатком r для цифры n_i. Если нам нужны две позиции цифр с противоположным остатком, то можно взять два таких минимальных индекса по очереди, и это даст правильный ответ.

Чтобы решить шестую подзадачу, нужно было полностью реализовать идею, описанную выше, не опираясь на неубывание цифр в числе. Время работы такого решения все еще будет равно $\mathcal{O}(k)$ — достаточно сделать константное количество полных проходов по числу. Заметим, что здесь мы сильно, хоть и неявно, опираемся на то, что n не содержит нулей. Однако комбинацией решения третьей и шестой подзадач уже можно было набрать 81 балл.

На самом деле, решение шестой подзадачи не верно для достаточно узкого класса чисел. Одно из таких чисел, например, равно 700222. Когда мы выбирали позиции, с которых нужно удалить цифры, мы не учли, что их удаление может породить ведущие нули. Кроме того, недостаточно применить алгоритм, описанный выше, и просто убрать ведущие нули, так как правильный ответ для такого числа равен 7002, а не 222.

Достаточно заметить, что ведущие нули такого рода в результате применения описанного алгоритма могут образоваться если число имеет вид

[`одна или две цифры, не кратные трем`] [`несколько нулей`] [`некоторый суффикс`].

Полное решение заключается в том, что нужно попробовать удалить нужное количество цифр не только из всего числа, но и отдельно только из его суффикса. Затем среди полученных результатов можно выбрать максимальный путем сравнения за $\mathcal{O}(k)$. Перед сравнением требуется избавится от возможного префикса из ведущих нулей.

Таким образом, несколько запусков алгоритма поиска и удаления одной цифры из числа будут работать за $\mathcal{O}(k)$, как и потенциальное сравнение нескольких возможных результатов, а значит и общее время работы равно $\mathcal{O}(k)$.

Задача C. Свободное перемещение

Автор задачи: Даниил Орешников, разработчик: Константин Бац

Задачу можно переформулировать как «ориентировать ребра неориентированного графа, чтобы максимизировать количество путей длины 2». Ключевой идеей для решения задачи являлось наблюдение, что количество путей длины 2 в графе в точности равно следующей сумме:

$$\sum_{v=1}^{n} \mathtt{deg}_{in}(v) \cdot \mathtt{deg}_{out}(v),$$

где $\mathtt{deg}_{in}(v)$ и $\mathtt{deg}_{out}$ — входящая и исходящая степени вершины v соответственно. Действительно, переберем «центральную» вершину пути, и заметим, что независимо друг от друга первую вершину можно выбрать $\mathtt{deg}_{in}$ способами, а третью — $\mathtt{deg}_{out}$ способами.

Для решения первой подзадачи можно было сделать полный перебор того, в какую сторону ориентировать каждое ребро, после чего посчитать для каждого способа указанную выше сумму. Такое решение работает за время $\mathcal{O}(2^n \cdot n)$.

Вторая подзадача решается даже без использования ключевой идеи. Граф, в котором степень каждой вершины не превосходит двух, разбивается на изолированные вершины, пути и циклы. Интуитивно очевидно, что количество путей длины 2 максимально тогда, когда ребра каждого пути и цикла ориентированы в одном направлении (то есть когда после ориентации ребер пути все еще являются путями, а циклы — циклами). Ориентировать ребра таким образом можно за один проход поиска в глубину за время $\mathcal{O}(n)$.

Для решения следующих подгрупп достаточно заметить, что $\mathtt{deg}_{in}(v) + \mathtt{deg}_{out}(v)$ равно $\mathtt{deg}(v)$, то есть степени вершины в исходном графе. При фиксированной сумма произведение двух величин тем больше, чем ближе эти две величины друг к другу. Таким образом, максимум достигается тогда, когда $|\mathtt{deg}_{in}(v) - \mathtt{deg})_{out}(v)|$ минимально, то есть равно 0 или 1 (в зависимости от четности степени вершины).

В третьей подзадаче есть конструктивный способ ориентировать ребра графа так, чтобы входящие степени были максимально близки к исходящим. Мысленно расположим все вершины графа по кругу и ориентируем ребра из вершины v в $\left\lfloor \frac{n}{2} \right\rfloor$ ближайших по направлению по часовой стрелке, тогда автоматически ребра из ближайших против часовой стрелки будут направлены из них в v. Останутся только ребра между противоположными по кругу вершинами, если n нечетно, их можно ориентировать в любую сторону. Время работы решения равно $\mathcal{O}(m)$.

Ориентировать дерево в четвертой подзадаче подобным образом тоже можно конструктивно: подвесим дерево за корень и будем спускаться от него рекурсивно вниз. Ориентируем ребра в половину детей вниз, а из половины детей — вверх. Если у корня нечетное число детей, ребро в оставшегося ребенка можно ориентировать в любую сторону.

Затем рекурсивно запустимся из всех детей. Если у текущей вершины четное число своих детей, ориентируем ровно по половине из них в каждую сторону. Если же нечетное — смотрим на ребро в саму текущую вершину сверху, если оно ориентировано в нее, то «лишнее» ребро надо ориентировать вниз в ребенка, и наоборот. Время работы решения равно $\mathcal{O}(n)$.

Для полного решения можно было разбить все вершины с нечетной степенью на пары, и провести в этих парах ребра. После такого действия все степени вершин станут четными, а значит в графе будет Эйлеров цикл — цикл, проходящий по каждому ребру ровно один раз. Запустим алгоритм поиска Эйлерова цикла (это просто поиск в глубину), и ориентируем ребра в том направлении, в котором по ним пройдем.

Теперь, благодаря свойствам Эйлерова цикла, выполняется $\mathtt{deg}_{in}(v) = \mathtt{deg}_{out}(v)$ для каждой вершины v. Если удалить те ребра, которые мы изначально добавили, степень каждой вершины изменится не более, чем на 1, таким образом входящие и исходящие степени каждой вершины максимально близки. Время работы решения, как и до этого — $\mathcal{O}(m)$.

Задача D. Подрыв ветряка

Автор задачи: Владимир Рябчун, разработчики: Даниил Орешников и Владимир Рябчун

Для решения первой подзадачи можно было не делать никаких наблюдений относительно того, в каком порядке надо отключать элементы — достаточно было перебрать все возможные перестановки и попробовать отключать их в таком порядке, пока к следующему элементу есть доступ. С помощью `std::next_permutation` решение можно было написать совсем коротко.

Вторая подзадача, аналогично, решалась полным перебором, однако здесь стоило применить динамическое программирования по подмножествам, $\texttt{dp}[mask]$ — минимальная стабильность, начиная с которой можно отключить в некотором порядке все элементы, закодированные маской $mask$. Пересчет динамики осуществлялся перебором первого элемента для отключения. Время работы решения — $\mathcal{O}(2^n \cdot n)$.

Для решения следующих подгрупп следовало сделать некоторые наблюдения относительно порядка отключения элементов.

Для начала заметим, что если какой-то конечной стабильности можно добиться, то ее также можно добиться, отключая сначала только элементы с положительными b_i, а затем — только с отрицательными. Действительно, если мы сначала можем отключить элемент с отрицательным b_i, а затем с положительным b_j, то $s \geqslant a_i$ и $s - |b_i| \geqslant a_j$. Из этого следует, что $s + b_j \geqslant a_i$ и $s \geqslant a_j$, а значит их можно с тем же результатом отключить в противоположном порядке.

Аналогичное наблюдение можно сделать и касательно порядка, в котором можно отключать «положительные» элементы — если это можно сделать в каком-то порядке, то можно сделать и порядке возрастания a_i. Доказательство еще проще — при их отключении стабильность только растет, значит если элемент с большим a_i можно было отключить раньше, его можно будет отключить и позже.

Это позволяет решить третью подзадачу аналогично задаче о рюкзаке: отсортируем все «положительные» элементы по возрастанию a_i и посчитаем $\texttt{dp}[t][x]$ — «можно ли получить стабильность x, отключая только элементы из числа первых t». Пересчет динамики будет следующий:

$$\texttt{dp}[t][x] = \begin{cases} \texttt{dp}[t-1][x] & \text{если } x - b_t < a_t \\ \texttt{dp}[t-1][x] \text{ or } \texttt{dp}[t-1][x-b_t] & \text{иначе.} \end{cases}$$

После чего достаточно найти минимальное такое S, которое можно получить с помощью «положительных» элементов, и при котором можно отключить единственный «отрицательный». Если итоговое значение после его отключения лучше изначальной стабильности, оно и будет ответом. Время работы решения — $\mathcal{O}\left(n \cdot \left(s + \sum |b_i|\right)\right)$.

Если все отрицательные b_i равны между собой, то можно сделать аналогичное наблюдение о том, что их можно применять в порядке уменьшения a_i и получить все то же множество возможных результатов. Доказательство полностью аналогично рассмотренным выше. В таком случае достаточно объединить идеи второй и третьей подзадач: посчитаем

- $\texttt{dp}^+[t][x]$ — можно ли получить стабильность x отключением каких-то из первых (по возрастанию a_i) t «положительных» элементов;
- $\texttt{dp}^-[t][x]$ — минимальная стабильность, которую можно получить из x только отключением каких-то из первых (по возрастанию a_i) t «отрицательных» элементов.

Пересчет такой динамики аналогичен, $\texttt{dp}^-[t][x]$ равно либо $\texttt{dp}^-[t-1][x]$, либо $\texttt{dp}^-[t-1][x+b_t]$, если $x \geqslant a_t$. Достаточно выбрать минимальное возможное значение. Время работы решения такое же, как и в предыдущей подгруппе.

В последних двух подгруппах отличие в том, что не всегда можно прийти к оптимальному ответу, отключая «отрицательные» элементы в порядке убываения a_i. В предпоследней подгруппе можно было исправить это, взяв динамику, описанную во второй подгруппе вместо пятой. Для полного решения же можно доказать, что достаточно отключать «отрицательные» элементы в порядке убывания $a_i + b_i$.

Действительно, пусть $a_i + b_i < a_j + b_j$ и верны неравенства

$$\begin{cases} s \geqslant a_i \\ s + b_i \geqslant a_j. \end{cases}$$

Из второго неравенства следует, что $s \geqslant a_j - b_i$, что больше $a_i - b_j$ по нашему предположению. А из этого как раз следует, что

$$\begin{cases} s \geqslant a_j & \text{так как } b_i < 0 \\ s + b_j \geqslant a_i, \end{cases}$$

а значит соответствующие элементы можно отключить в обратном порядке. Дальше достаточно применить решение из пятой подгруппы, только отсортировать «отрицательные» элементы в порядке возрастания $a_i + b_i$. Время работы решения то же самое.