Задача A. Обратный отсчет

Мы можем перебирать элементы и отслеживать количество последовательных элементов так, чтобы следующий элемент был на единицу меньше предыдущего. Для этого мы ведем счетчик. Если текущий элемент на единицу меньше предыдущего, мы увеличиваем этот счетчик на 1. В противном случае мы сбрасываем счетчик в 0. Если текущий элемент равен 1, а наш счетчик не менее $k-1$, мы знаем, что текущий элемент является концом обратного отсчета k. В этом случае мы можем увеличить счетчик ответов на 1.

Этот подход работает, поскольку любая пара подотрезков k-отсчета не пересекается. Это решение работает за $O(n)$.

Задача B. Разкраченность последовательности

В этой задаче нам нужно отсортировать массив x и найти максимальную разницу каких-либо двух соседних элементов, но сортировать каждый массив за $\mathcal{O}(n \log n)$ мы не можем, так как это будет долго. Есть два подхода к решению задачи:

1. Блочная сортировка (Bucket sort). Мы знаем, что значения в массиве не больше 10^9, поэтому давайте будем представлять каждое значение в виде пары чисел. Пусть $k = \lceil \sqrt{10^9} \rceil$. Тогда представим каждое x_i как пару чисел (a_i, b_i), где $a_i = \lfloor \frac{a_i}{k} \rfloor$ и $b_i = a_i \bmod k$. Тогда отсортировать массив x_i это то же самое, что и отсортировать массив пар (a_i, b_i). Сразу же отметим, что при таком подходе все a_i и b_i не превосходят k.

   Далее проведем сортировку пар в два этапа:

   - Для начала отсортируем все пары (a_i, b_i) по неубыванию значения b_i. Мы можем это сделать за $\mathcal{O}(n+k)$ с помощью сортировки подсчётом.
   - Далее отсортируем наш новый полученный массив по неубыванию значения a_i. Это мы также можем сделать за $\mathcal{O}(n+k)$ с помощью сортировки подсчётом.

   Нам была важна первая итерация сортировки, так как сортировка подсчётом является стабильной сортировкой. Напомним, что сортировка называется стабильной, если равные элементы сохраняют свой относительный порядок.

   Рассмотрим процесс этой сортировки на примере $(2,1), (1,2), (1,1), (2,2)$:

   - Сначала мы сортируем пары по второму значению и получаем массив $(2,1), (1,1), (1,2), (2,2)$.
   - Далее мы сортируем пары по первому значению и получаем массив $(1,1), (1,2), (2,1), (2,2)$.

   Решение можно реализовать за $\mathcal{O}(n+k)$ для одного набора входных данных.

2. Пусть mn — минимальный элемент в массиве x; mx — максимальный элемент. Тогда мы точно можем сказать, что максимальная разница будет хотя бы $b = \lceil \frac{mx-mn}{n-1} \rceil$. Давайте все x_i разделим на следующие блоки по значениям: $[mn, mn+b-1]$, $[mn+b, mn+2 \cdot b-1]$, и так далее. Всего у нас получится не более n блоков и элемент x_i попадёт в блок с номером $\lfloor \frac{x_i-mn}{b} \rfloor$.

   Так как мы уже сделали оценку на ответ и разделили числа соответствующим образом на блоки, то внутри одного блока не будет пары чисел с разницей хотя бы b. Значит, нам не нужно рассматривать и сортировать числа внутри блоков, нам достаточно рассматривать только числа между блоками, а для этого нам нужно знать только минимальное и максимальное значение в каждом блоке.

   Рассмотрим это решение на примере массива $x = [32, 1, 31, 74, 92, 31, 52]$. В этом случае $mn = 1$, $mx = 92$ и мы можем сразу сделать оценку на ответ, что он хотя бы $b = \lceil \frac{92-1}{6} \rceil = 16$. Тогда числа поделятся на следующие блоки: $[[1], [32, 31, 31], [], [52], [74], [92]]$. Мы понимаем, что внутри одного блока между числами разница меньше b, поэтому такие пары чисел мы не будем

рассматривать. Мы будем рассматривать только числа между блоками, и в каждом блоке оставим только минимальное и максимальное значения. Тогда нам нужно будет рассмотреть пары чисел 1 и 31, 32 и 52, 52 и 74, 74 и 92.

Такое решение можно реализовать за $\mathcal{O}(n)$ на один набор входных данных.

Задача C. Прыжки по кругу

Чтобы решить задачу, сделаем следующие наблюдения:

- Если применить операцию $b = n - 1$, то перестановка s не изменится. Сам массив s, конечно, сдвинется, но с точки зрения задачи на круге он не изменится.
- Если применить операцию $b = n$, то элемент со значением n сдвинется на один вперёд, а все остальные элементы не поменяют свой относительный порядок.

Давайте будем перебирать значения x в порядке убывания от $n - 3$ до 1 и делать операции таким образом, чтобы после рассмотрения значения x числа $x, x + 1, \ldots, n - 1$ были в нужном нам порядке относительно друг друга (в таком порядке, как они расположены в перестановке t). Мы будем перебирать x, начиная с $n - 3$, так как элементы $n - 2$ и $n - 1$ в любом случае находятся в нужном нам порядке (не забываем, что мы решаем задачу на круге). Когда мы сделаем так, чтобы элементы $1, 2, \ldots, n - 1$ находились в нужном нам порядке, мы сможем за не более чем n операций поставить число n в нужную нам позицию.

Пусть элементы $x+1, x+2, \ldots, n-1$ уже находятся в нужном для нас порядке. Научимся ставить на нужную нам позицию число x. Для этого нам нужно определить элемент, перед которым должен стоять x. На самом деле это такой элемент $y \in [x + 1, n - 1]$, что расстояние от числа x до числа y в перестановке t минимально. Здесь мы рассматриваем ориентированное расстояние. Другими словами, от элемента на позиции p на расстоянии один находится только элемент на позиции $p \bmod n + 1$.

Пусть мы нашли нужное нам значение y. Теперь мы должны сделать какие-то операции, чтобы элемент со значением x стоял перед элементом со значением y в перестановке s. Пусть d — расстояние от элемента со значением x до элемента со значением y в перестановке s. Далее нам нужно рассмотреть следующие случаи:

- Пусть $g = \gcd(x, n-1)$. Мы знаем, что $n-1$ прыжок вперёд не меняет перестановку s, а элемент со значением x делает за одну операцию ровно x прыжков. Другими словами, с помощью некоторого количества операций с $b = x$ мы можем переместить элемент x на $g, 2 \cdot g, 3 \cdot g, \ldots$ прыжков вперёд.
- Если $(d - 1)$ делится на g, то нам достаточно проводить операции с $b = x$, и когда-то элемент x окажется перед элементом y в перестановке s. Мы берём значение $d - 1$, так как текущее расстояние между элементами d, а нам нужно, чтобы они находились на расстоянии 1.
- Если $(d-1)$ не делится на g, то нам нужно делать вспомогательные операции, чтобы перейти в случай, когда $(d-1)$ будет делиться на g. Для этого мы будем использовать вспомогательный элемент со значением n, пока не окажемся в нужном нам случае. Алгоритм действий будет следующий:
  - Применяем операции для $b = n$ до тех пор, пока элемент со значением n не окажется перед элементом со значением x в перестановке s.
  - Применяем операцию для $b = x$.

  Можно показать, что с помощью такого алгоритма мы постепенно уменьшаем остаток от деления числа $(d - 1)$ на g. Удобнее всего это увидеть на примерах.

Решение можно реализовать за $\mathcal{O}(n^3)$ или $\mathcal{O}(n^4)$ в зависимости от реализации.

Задача D. След латинского квадрата

У задачи есть много разных решений, в том числе и с использованием паросочетаний. Но в таких решениях нужно многие вещи доказывать или понимать интуитивно, поэтому мы остановимся на конструктивном решении. Сделаем следующие наблюдения:

- Ответа не существует, если $k = n + 1$ или $k = n^2 - 1$. Это два симметричных случая, докажем, почему не существует ответа для $k = n + 1$. Получить значение $n + 1$ с помощью n положительных слагаемых можно только одним способом: $n + 1 = 1 \cdot (n - 1) + 2 \cdot 1$. Но если у нас на диагонали некоторое значение x встречается хотя бы $n - 1$ раз, то оно должно встречаться ровно n раз. Это связано с тем, что значения в строках и столбцах должны быть различными.
- Для всех остальных значений k ответ существует. И все остальные значения k можно разделить на три группы:
  1. $k = i \cdot n$ для некоторого целого числа $1 \leqslant i \leqslant n$.
  2. $k = i \cdot (n - 2) + 2 \cdot j$ для некоторых целых чисел $1 \leqslant i \neq j \leqslant n$.
  3. $k = i \cdot (n - 2) + j + z$ для некоторых различных целых значений $1 \leqslant i, j, z \leqslant n$.

  Этих случаев хватит, чтобы представить k в виде суммы n положительных слагаемых, где $n \leqslant k \leqslant n^2$, $k \neq n + 1$ и $k \neq n^2 - 1$.

Когда мы представим значение k в каком-нибудь одном из трёх вышеперечисленных способов, мы сможем построить ответ конструктивно. На самом деле эти способы представить значение k в виде суммы n слагаемых — это способы зафиксировать диагональ естественного латинского квадрата. Мы не будем подробно описывать конструктивное решение и лишь покажем его на частных примерах.

3	1	2	4	5	6
6	3	1	2	4	5
5	6	3	1	2	4
4	5	6	3	1	2
2	4	5	6	3	1
1	2	4	5	6	3

2	6	1	3	4	5
5	2	6	1	3	4
4	5	2	6	1	3
6	3	4	2	5	1
1	4	3	5	6	2
3	1	5	4	2	6

2	5	1	3	4	6
6	2	5	1	3	4
4	6	2	5	1	3
3	4	6	2	5	1
5	1	3	4	6	2
1	3	4	6	2	5

Разберём три примера выше:

- Левый пример для $n = 6$ и $k = 18$. Мы представляем $k = 3 \cdot 6$, то есть расставляем на диагонали все тройки. Далее мы произвольно выписываем оставшиеся числа в первой строке, а каждая следующая строка матрицы — это сдвинутая вправо предыдущая строка матрицы.
- Центральный пример для $n = 6$ и $k = 20$. Мы представляем $k = 2 \cdot (n - 2) + 6 \cdot 2$, то есть расставляем на диагонали четыре двойки и две шестёрки. Жёлтым и красным цветом отмечены ячейки с двойками и шестёрками соответственно. В первых $n - 2$ строках остальные числа заполняются также с помощью сдвига, а последние две строки матрицы нужно аккуратно рассмотреть отдельно.

- Правый пример для $n = 6$ и $k = 19$. Мы представляем $k = 2 \cdot (n-2) + 5 + 6$, то есть расставляем на диагонали четыре двойки, одну пятёрку и одну шестёрку. Если хорошо расставить эти три значения, то оставшиеся числа можно расставлять просто с помощью сдвига.

Задача E. Остаток суммы равен длине

В этой задаче нам нужно посчитать количество подотрезков, для которых $(a_l + a_{l+1} + \ldots + a_r) \bmod k = r - l + 1$. Сразу же вспомним, что сумму на отрезке нам удобнее считать с помощью префиксных сумм. Пусть $pref_i = a_1 + a_2 + \ldots + a_i = pref_{i-1} + a_i$. Тогда мы можем переформулировать задачу следующим образом: нам нужно посчитать количество пар $0 \leqslant l < r \leqslant n$, таких что $(pref_r - pref_l) \bmod k = r - l$. Далее нам нужно более внимательно рассмотреть это условие равенства:

- Это условие выполняется, только если $r - l < k$.
- Это условие равенства равносильно $(pref_r - r) \bmod k = (pref_l - l) \bmod k$.

Пусть $b_i = (pref_i - i) \bmod k$. Тогда наша задача посчитать количество пар $0 \leqslant l < r \leqslant n$, таких что $r - l < k$ и $b_l = b_r$. Эту задачу мы уже можем решать с помощью словаря (map). Для этого будем перебирать все правые границы $r \in [1, n]$, а в словаре поддерживать все значения b_l (и их количество), такие что $r - l < k$. Решение можно реализовать за $\mathcal{O}(n \log n)$.

Задача F. Нанопроцессор

Будем решать задачу жадно, перебирая строки сетки в порядке возрастания. Так как мы знаем количество ботвизаторов в i-й строке, то мы знаем, сколько ячеек в этой строке принадлежат расквантизаторам: $c_i = m - a_i$. Ячейки, которые принадлежат расквантизаторам, можно разделить на две категории:

- Не верхние ячейки расквантизатора. Эти ячейки уже обязаны быть частью расквантизаторов, так как мы их начали ранее.
- Верхние ячейки расквантизатора.

Пусть в i-й строке есть p ячеек, которые уже точно будут принадлежать расквантизаторам. Если $p > c_i$, то мы получаем противоречие, и ответ будет равен «`inconsistent`». Если $p \leqslant c_i$, то нам нужно выбрать ещё $c_i - p$ ячеек, которые будут верхними ячейками расквантизаторов. Какие ячейки нам стоит выбрать?

Во-первых, мы будем выбирать только среди тех ячеек, которые ещё не принадлежат каким-либо расквантизаторам. Во-вторых, если обозначить как b_j — количество расквантизаторов, которые ещё **необходимо поставить** в j-м столбце, то нам выгодно выбрать такие $c_i - p$ ячеек, что их значения b_j максимальны. Если где-то в процессе такого жадного решения мы получим противоречие: например, не все расквантизаторы установили в каком-либо столбце, какой-либо расквантизатор не поместился полностью в матрице или что-либо другое, то ответ будет «`inconsistent`».

Решение можно реализовать за $\mathcal{O}(n \cdot m \log m)$.

Задача G. Четкая лаборатория

Будем решать задачу методом динамического программирования. Пусть $dp[l][r]$ — минимальное время, чтобы мы сделали $y = s_l s_{l+1} \ldots s_r$. В таком случае ответ на задачу будет равен $dp[1][n] - b$, так как в нашей задаче нам нужно сделать $x = s$ и нам не придётся делать последнюю операцию типа **B**.

Нам удобнее будет считать эту динамику вперёд, перебирая состояния (l, r) в порядке возрастания величины $r - l$. База динамики будет следующей: $dp[l][r] = a + b$. Рассмотрим все необходимые переходы из состояния (l, r):

- Мы можем сделать следующее обновление: $dp[l][r+1] = \min(dp[l][r+1], dp[l][r] + a)$. Для этого мы можем сделать x равным $s_l s_{l+1} \dots s_r$, дописать символ s_{r+1} за a единиц времени и применить операцию типа **B**.

- Мы можем сделать следующее обновление: $dp[l-1][r] = \min(dp[l-1][r], dp[l][r] + a)$. В процессе получения $y = s_l s_{l+1} \dots s_r$, мы в какой-то момент получили для начала $x = s_l s_{l+1} \dots s_r$. Чтобы получить нужную нам строку x, сразу после последнего использования операции типа **B** мы можем добавить символ s_{l-1} в начало (после операции типа **B** строка x пустая, поэтому символ s_{l-1} будет первым) и сделать те же самые операции, что и раньше, чтобы получить $x = s_{l-1} s_l \dots s_r$.

- В двух переходах выше мы не рассматривали возможность применения операции типа **C**. Рассмотрим такие варианты перехода. Мы будем переходить из состояния (l, r) в некоторое состояние (l, r'). При этом теперь мы хотим как можно больше использовать операцию типа **C**, а строка y сейчас равна $s_l s_{l+1} \dots s_r$.

  Пусть d — максимальное количество непересекающихся вхождений строки $s_l s_{l+1} \dots s_r$ в строку $s_l s_{l+1} \dots s_{r'}$. Тогда мы можем сделать следующее обновление: $dp[l][r'] = \min(dp[l][r'], dp[l][r] + d \cdot c + a \cdot (r' - l + 1 - d \cdot (r - l + 1)) + b)$. Другими словами, мы сначала за $dp[l][r]$ единиц времени делаем x пустым и y равным $s_l s_{l+1} \dots s_r$, затем за $d \cdot c + a \cdot (r' - l + 1 - d \cdot (r - l + 1))$ делаем строку x равной $s_l s_{l+1} \dots s_{r'}$, а затем применяем операцию типа **B**. В этих формулах величина $r' - l + 1 - d \cdot (r - l + 1)$ равна количеству символов, которые мы будем добавлять отдельно с помощью операции типа **A**.

  Нам нужно научиться делать такой пересчёт достаточно быстро. Для этого нам нужны следующие два наблюдения:

  - Нам не нужно обновлять динамику для всех значений r', так как у нас есть переход из (l, r) в $(l, r+1)$ за a единиц времени. Нам достаточно обновлять динамику для таких значений r', что для них значения d различны, и значения r' минимальны.
  - Для этого нам нужно находить самые левые непересекающиеся вхождения строки $s_l s_{l+1} \dots s_r$ в строку $s_l s_{l+1} \dots s_n$. Давайте для этого предпосчитаем следующие динамики:
    - $pref[l][r]$ — наибольший общий префикс строк $s_l s_{l+1} \dots s_n$ и $s_r s_{r+1} \dots s_n$. Заметим, что если $s_l \neq s_r$, то $pref[l][r] = 0$, а иначе $pref[l][r] = 1 + pref[l+1][r+1]$.
    - $next[l][r]$ — наименьший индекс $i > r$, такой что $s_l s_{l+1} \dots s_r = s_i s_{i+1} \dots s_{i+r-l}$. Другими словами, $next[l][r]$ — наименьший индекс $i > r$, что $pref[l][i] \geqslant r - l + 1$. Предпосчитать массив $next_l$ для фиксированного значения l можно за $\mathcal{O}(n)$ с помощью указателя.

  Тогда нам нужно рассматривать только следующие значения r': $r_1 = next[l][r] + r - l$, $r_2 = next[r_1 - (r - l)][r_1]$ и так далее.

Сделаем замечание, почему мы не рассматриваем переходы с помощью операций типа **C** из (l, r) в (l', r'), где $l' \leqslant l$ и $r' \geqslant r$. Дело в том, что если у нас есть несколько одинаковых подстрок, то мы можем считать, что помещали в буфер обмена самую левую из них.

Оценим время работы такого решения. Массивы $pref$ и $next$ мы строим за $\mathcal{O}(n^2)$. Для состояния (l, r) мы рассматриваем не более $\mathcal{O}(\frac{n}{r-l+1})$ переходов. Если зафиксировать значение l, то суммарно мы рассматриваем не более $\mathcal{O}(\frac{n}{1} + \frac{n}{2} + \ldots + \frac{n}{n-l+1}) = \mathcal{O}(n \log n)$ переходов. Решение можно реализовать за $\mathcal{O}(n^2 \log n)$.

Задача H. Panely

Так как каждый панелер должен быть наставником ровно одного другого панелера и граф наставничества должен быть связным, то наша задача сделать граф-цикл на n вершинах. Будем решать задачу жадно и поддерживать следующие значения:

- Будем поддерживать множество панелеров, которые ещё не являются наставниками. Изначально это множество содержит все целые числа от 1 до n.

- Заметим, что если в хорошем распределении мы зафиксируем некоторый префикс наставников (префикс, длина которого меньше n), то граф наставничества будет состоять из цепочек. От каждой цепочки нам важно знать только первого и последнего панелера в ней, поэтому будем поддерживать такие пары. Изначально это можно рассматривать как пары $b_i = \{i, i\}$ — мы ещё не выбрали никому наставников, поэтому никаких связей нет и каждая цепочка состоит ровно из одного панелера i.

Теперь будем решать задачу жадно, пусть мы уже определили наставников для первых $i - 1$ панелеров, научимся определять наставника для i-го панелера. Сразу отметим, что для последнего панелера (панелера с номером n) мы выберем наставника однозначно. Для других панелеров нам нужно рассмотреть следующие случаи:

- Для начала мы постараемся сохранить в качестве наставника панелера a_i. В каком случае это возможно? Мы можем выбрать панелера a_i наставником, если его номер есть в множестве, а также $b_i \neq \{i, a_i\}$. Второе условие для нас важно, так как нам нельзя образовывать изолированные группы.

- Если предыдущий пункт не выполняется, тогда мы хотим выбрать в качестве наставника панелера x, что его номер минимален (минимальный элемент множества, который мы можем узнать за $\mathcal{O}(1)$). Но нам всё также нужно проверить, что $b_i \neq \{i, x\}$, так как нам нельзя образовывать изолированные группы. Если такой панелер x нам также не подходит, то мы сможем взять в качестве наставника панелера со вторым минимальным значением в множестве.

Пусть мы выбрали x — наставник панелера i. Тогда нам нужно пересчитать наши значения.

- Из множества потенциальных наставников нам нужно удалить значение x.

- Так как x является наставником i, то нам нужно объединить две цепочки в одну. Теперь крайними панелерами новой цепочки будут панелеры $c = b_x.s$ и $d = a_x.s$ (здесь мы с помощью $.s$ обозначаем второй элемент пары). Таким образом мы должны сказать, что $b_c = \{c, d\}$ и $b_d = \{d, c\}$.

  Здесь сразу же сделаем замечание, что мы поддерживаем корректные значения b только для тех панелеров, у которых ещё нет либо наставника, либо подопечного. Для других панелеров нам уже не нужно поддерживать эту информацию, так как мы её не будем использовать.

Решение можно реализовать за $\mathcal{O}(n \log n)$.

Задача I. Квартира для Дениса

Пусть d — ответ на задачу. Давайте подумаем, почему мы не смогли увеличить ответ на какое-нибудь маленькое значение eps. Есть два варианта:

1. Мы не можем увеличить ответ на задачу, так как будет существовать друг i, такой что не существует точки на расстоянии не более r_i от точки (x_i, y_i), которая находится на расстоянии хотя бы $d + eps$ от точки $(0, 0)$. Другими словами, окружность с центром в точке $(0, 0)$ и радиусом d полностью содержит в себе окружность с центром в точке (x_i, y_i) и радиусом r_i.

2. Существует пара друзей i и j, что область пересечения их окружностей (мы можем рассматривать каждого друга как окружность с центром в его квартире и радиусом r) полностью содержится в окружности с центром в точке $(0, 0)$ и радиусом d.

   Здесь дополнительно нужно сделать ещё несколько наблюдений. Если для пары друзей верно, что окружность одного вложена в окружность другого, то мы можем считать, что мы находимся внутри первого случая. Если же нет вложенности, то окружности пересекаются в

каких-то двух точках (может быть пересечение и по одной точке, но тогда ответ определяется однозначно).

Утверждение: если точки пересечения двух окружностей полностью лежат в окружности с центром в точке $(0, 0)$ и радиусом d, то и вся область пересечения этих двух окружностей лежит внутри данной окружности.

Из наблюдений выше мы можем составить список кандидатов точек, где Денис может поселиться. Будут две категории точек:

1. Наиболее удалённая от точки $(0, 0)$ точка окружности с центром в (x_i, y_i) и радиусом r_i. Нам будет выгодно взять такую точку (a, b), что вектора (a, b) и (x_i, y_i) коллинеарны, точка (a, b) лежит на окружности и наиболее удалена от точки $(0, 0)$. Другими словами, нам нужно удлинить вектор (x_i, y_i) на r_i и мы получим точку (a, b). Здесь нужно аккуратно рассмотреть крайний случай, когда $(x_i, y_i) = (0, 0)$.

2. Точки, которые являются пересечением какой-либо пары окружностей.

Каждую точку-кандидат мы можем проверить на корректность за $\mathcal{O}(n)$ (нам нужно будет проверить, что точка-кандидат подходит для всех друзей). Так как всего точек-кандидатов будет $\mathcal{O}(n^2)$, то мы можем реализовать решение за $\mathcal{O}(n^3)$.

Задача J. Красота расположения книг

Будем жадно набирать ответ. Для начала мы хотим максимизировать первую цифру, то есть a_1. И здесь у нас есть два варианта:

- Существует цифра $a_j > a_1$. В этом случае нам обязательно нужно применить операцию с цифрой a_1, так как нам важно её максимизировать. Наиболее выгодно применить эту операцию с таким значением a_j, что a_j максимально, а среди всех таких вариантов выбрать индекс j, нас будет интересовать самый правый.

  Удобнее увидеть эту идею на примере $a = [1, 3, 2, 3]$. Нам невыгодно менять местами a_1 и a_3, так как тогда мы получим массив $a = [2, 3, 1, 3]$. С другой стороны, нам наиболее выгодно поменять местами a_1 и a_4, нежели a_1 и a_2: в первом случае мы получим массив $[3, 3, 2, 1]$, а во втором $[3, 1, 2, 3]$.

- Такой цифры a_j не существует. Тогда не будем применять операцию, перейдём ко второй цифре и попытаемся её максимизировать аналогичными рассуждениями.

Чтобы реализовать эту часть решения, можно предпосчитать массив суффиксных максимумов для массива a. Если в процессе этого алгоритма мы не применили операцию, то на самом деле нам дан невозрастающий массив. Другими словами, $a_1 \geqslant a_2 \geqslant \ldots \geqslant a_n$. Рассмотрим этот случай отдельно:

- Если существует $1 \leqslant i < n$, такое что $a_i = a_{i+1}$, то применим операцию между этими индексами. Тем самым ответом на задачу будет исходный массив a.

- Если такого индекса i не существует, то выгодно поменять местами элементы a_{n-1} и a_n.

Решение можно реализовать за $\mathcal{O}(n)$.

Задача L. Доставка по кампусу

Чтобы решить эту задачу, нам нужно вспомнить решения двух базовых её подзадач:

- Есть неориентированный взвешенный граф, веса рёбер неотрицательны, в нём нужно найти кратчайшие расстояния. Эту подзадачу мы можем решать с помощью алгоритма Дейкстры.

- Есть ациклический ориентированный взвешенный граф, веса рёбер могут быть произвольными, и нам нужно найти кратчайшие расстояния. Эту подзадачу мы решаем с помощью топологической сортировки. Так как граф ациклический, то мы можем её найти и рассматривать все вершины и рёбра из них в этом порядке.

В нашей задаче следующие условия:

- Все наземные переходы двусторонние, и их стоимости неотрицательные. Это означает, что если рассматривать отдельно какую-либо компоненту связности по наземным переходам, то мы сможем воспользоваться алгоритмом Дейкстры.
- Стоимости экспресс-туннелей могут быть произвольными, но они соединяют вершины из разных компонент сильной связности, а для обработки этого случая мы можем использовать топологическую сортировку.

Общее решение будет выглядеть следующим образом:

- Выделим все компоненты связности по наземным переходам.
- Далее проведём экспресс-туннели между полученными компонентами связности и найдём топологическую сортировку этих компонент.
- Будем рассматривать все компоненты в порядке топологической сортировки и решать задачу для каждой из них с помощью алгоритма Дейкстры. Очень важно, что решая задачу внутри одной компоненты, мы рассматриваем только наземные переходы, то есть рёбра внутри компоненты.
- После обработки компоненты связности мы рассматриваем все экспресс-туннели, ведущие из неё, и обновляем расстояния до соответствующих вершин. Здесь отметим, что мы не добавляем такие переходы в алгоритм Дейкстры — мы каждую вершину рассматриваем в алгоритме Дейкстры только тогда, когда рассматриваем её компоненту связности.

Решение можно реализовать за $\mathcal{O}((r+p)\log n)$.

Задача M. Пробежка студентов ФПМИ

Для каждого студента мы знаем, с какой позиции он начнёт забег и в какой позиции его закончит. Обозначим их как $l_i = x_i$ и $r_i = x_i + t \cdot v_i$. Два студента i и j не могут бежать по одной дорожке, если выполняется одно из следующих двух условий:

- $l_i \leqslant l_j$ и $r_j \leqslant r_i$.
- $l_j \leqslant l_i$ и $r_i \leqslant r_j$.

Другими словами, два студента не могут бежать по одной дорожке, если отрезок забега одного вложен в отрезок забега другого. На самом деле нам нужно найти наибольшую вложенность данных нам отрезков — это и будет ответ на задачу. Но мы рассмотрим другое жадное решение.

Будем рассматривать студентов в порядке увеличения их стартовой позиции. По условию задачи все x_i различны, поэтому нам не нужно задумываться о случае равенства значений l_i. В некотором множестве будем поддерживать информацию об уже имеющихся у нас дорожках. Изначально это множество пустое. Каждую дорожку мы можем описать некоторым значением r_i — самой правой позицией, в которой закончит забег участник этой дорожки.

Пусть мы сейчас рассматриваем i-го студента. Для начала мы попробуем его определить на уже имеющуюся дорожку. На какую дорожку мы его можем определить? Мы можем его определить на такую дорожку, что её значение r_j меньше чем r_i. Если значение r_j будет хотя бы r_i, то тогда образуется вложенность — у студента, бегущего по этой дорожке, значение l_j меньше значения l_i, а значение r_j будет не меньше значения r_i. Среди всех подходящих дорожек нам выгодно будет выбрать такую дорожку, что её значение r_j максимально.

Другими словами, нам нужно найти в множестве такое значение r_j, что $r_j < r_i$ и r_j максимально. Это мы можем сделать с помощью встроенного метода lower_bound, чтобы найти подходящее значение r_j (нужно не забыть вычесть единицу из найденного итератора, так как lower_bound находит наименьший больше либо равный элемент). Если такое значение r_j существует, то мы его удаляем из множества и добавляем значение r_i (тем самым мы обновляем правую границу для этой дорожки), а если такого значения не существует, то для данного студента нужно создать новую дорожку и просто добавить в множество значение r_i.

Ответ на задачу будет равен размеру получившегося множества. Решение можно реализовать за $\mathcal{O}(n \log n)$.

Задача N. Цензура

Обозначим за $T = |t_1| + |t_2| + \ldots + |t_n|$. Другими словами, T — суммарная длина всех слов t_i. В нашей задаче $T \leqslant 10^5$.

Заметим, что **различных** длин среди строк $t_1, t_2, \ldots, t_n$ не больше $\mathcal{O}(\sqrt{T})$. Для каждой строки t_i посчитаем её хеш и для каждой длины len, что в наборе t есть хотя бы одна строка длины len, запомним хеши таких строк. Тогда далее мы можем решать задачу жадно с помощью стека:

- Будем перебирать символы строки s слева направо и поддерживать наш текущий стек (строку s, которая получилась бы, если бы мы решали задачу для заданного префикса). Для данного стека мы будем поддерживать хеши всех префиксов.
- Добавить очередной символ мы можем за $\mathcal{O}(1)$ — для этого достаточно добавить в конец стека символ, и у нас появится ещё одно значение префиксного хеша.

  Что может произойти после добавления нового символа? Если у строки, которая была до добавления очередного символа, не было подстроки t_i, то после добавления может появиться ровно одна такая строка. Давайте переберём длину этой строки len (напомним, что различных длин среди t_i не больше $\mathcal{O}(\sqrt{T})$), посчитаем хеш последних len символов нашего текущего стека, и если в нашем наборе есть строка длины len с таким хешом, то мы должны будем удалить из стека последние len символов. Также мы должны будем удалить последние len значений нашей префиксной хеш-функции.

Решение можно реализовать за $\mathcal{O}(|s|\sqrt{T})$.