Задача 1. Видеонаблюдение

Автор: Максим Деб Натх
Разработчики: Максим Деб Натх, Тихон Евтеев

Рассмотрим для начала одномерную задачу ($h = 1$): пусть у нас есть горизонтальная полоса длины w, на которой отмечены ячейки с координатами c_i. Отсортируем координаты по возрастанию. Можно перебирать сдвиги этой полосы и выбрать среди них правильный ответ. Однако существует более эффективное решение:

Рассмотрим пару соседних занятых ячеек c_i и c_{i+1}. Если сдвинуть поле так, чтобы все свободные ячейки между рассматриваемой парой оказались либо в начале, либо в конце полосы соответственно, оставшиеся с полосе занятые ячейки будут содержаться в «подполосе» длины $w - c_{i+1} + c_i + 1$.

Чтобы добиться такого свига, нам потребуется либо c_i сдвигов влево, либо $w - c_{i+1} + 1$ сдвигов вправо. Остается не забыть, что за 0 сдвигов можно получить подполосу длиной $c_n - c_1 + 1$.

Переберем пары соседних элементов, и среди тех, расстояние между которыми максимально, найдем пару с минимальным сдвигом согласно формулам выше выше. Таким образом, мы решили одномерную задачу.

Для полного решения остается заметить, что задачи по двум координатам независимы. Ширина покрывающего многоугольника не зависит от сдвигов вверх и вниз, а длина — от сдвигов влево и вправо. Так что получим ответы по двум измерениям, перемножим полученные длины, чтобы получить площадь прямоугольника («компактность»), и сложим количества действий по каждому измерению.

Задача 2. Тайное послание

Автор: Денис Кириенко
Разработчики: Алеся Иванова, Николай Будин

В подзадаче 1 значение $k_i = 1$, то есть $|T| = 1$. Тогда для передачи сообщения из единственного числа t_1 можно передать число $(t_1 + 1) \bmod n$. Здесь и далее под $\bmod n$ подразумевается операция взятия остатка от деления на n для тех чисел, которые больше n. Аналогичную идею (прибавить к каждому числу по 1) нужно применить и в подзадаче 5, где для каждого числа t_i гарантируется, что $t_i + 1$ не входит в T.

В подзадаче 2 необходимо передать два числа t_1 и t_2, аналогично можно передать числа $(t_1 + 1) \bmod n$ и $(t_2 + 1) \bmod n$, если числа t_1 и t_2 не соседние (с учётом зацикливания, т.е. следующим после числа n будем считать число 1). Если же числа t_1 и t_2 соседние (с учётом зацикливания, числа n и 1 также считаем соседними), то можно взять числа $(t_1+2) \bmod n$ и $(t_2+2) \bmod n$. Процесс дешифрования сообщения устроен так же: если получены не соседние числа, то нужно вычесть из каждого из них 1, иначе нужно вычесть 2.

В подзадаче 3 значение $k = n/2$, поэтому множество R будет являться дополнением множества T.

В подзадаче 4 нужно рассмотреть случай $k = 3$ при $n = 7$. Таких подмножеств будет 35, можно изучить структуру таких множеств и сопоставить одному множеству другое, остальные значения n и k сводятся к частным случаям $k = 1$, $k = 2$ и $k = n/2$.

В подзадаче 6 отрезок между минимальным и максимальным значением t_i меньше, чем $n/2$, поэтому множество T можно сдвинуть, прибавив ко всем его элементам значение $t_k - t_1 + 1$, то есть число t_i переходит в число $(t_i + t_k - t_1 + 1) \bmod n$.

Идеи, возникшие при анализе частных случаев задачи, позволяют построить общее решение. Давайте для каждого передаваемого числа t_i попробуем добавить следующее за ним число, то есть $t_i + 1$. Но если число $t_i + 1$ также входит в множество T, то перейдём к рассмотрению числа $t_i + 2$, но тогда нам нужно будет добавить в множество R уже два следующих числа. Таким образом, мы пытаемся построить множество R из k элементов так, чтобы между элементами множества T и элементами множества R, в которые они перешли, не было чисел, не попавших ни в T, ни в R.

Это можно реализовать следующим образом. Пусть переменная *count* равна количеству элементов, которое нам необходимо добавить к множеству R. Рассматриваем числа последовательно. Если текущее рассматриваемое число входит в множество T, то увеличиваем *count* на 1. Если текущее

рассматриваемое число не входит в множество T, то добавляем его в множество R и уменьшаем *count* на 1. Если после рассмотрения числа n оказалось, что $count > 0$, то необходимо добавить *count* минимальных чисел (начиная с 1), которые не входят ни в T, ни в построенное R.

Количество набираемых таким решением баллов зависит от эффективности реализации данного алгоритма. Например, если последовательно рассматривать все числа от 1 до n, и про каждое число от 1 до n принимать решение в зависимости от значения *count*, то получится решение сложности $O(N)$, которое наберёт 86 баллов за подзадачи 1–10.

В полном решении предполагается сложность $O(K)$ или $O(K \log K)$, для чего необходимо не рассматривать все числа от 1 до n, а рассматривать только числа из множества T. Если $count = 0$, то в качестве текущего рассматриваемого значения нужно взять следующее число из множества T, а если $count > 0$, то увеличиваем текущее число на 1. То есть если $count = 0$ (мы зашифровали все близкие числа одной группы множества T), будем сразу же переходить к следующему числу из множества T. Если после рассмотрения числа n значение $count > 0$, то при добавлении *count* маленьких чисел в сложности решения может возникнуть множитель $\log K$, если использовать контейнер set для хранения множеств T и R. Возможно реализовать решение и за линейную сложность $O(K)$, но это не требовалось в этой задаче.

В таком решении можно увидеть аналогии с правильными скобочными последовательностями, где числа множества T соответствуют открывающим скобкам и увеличивают баланс, а числа множества R уменьшают баланс и являются закрывающими скобками. Также структуру ответа можно представить, как хеш-таблицу с открытой адресацией размера n, где k элементов уже заняты элементами множества T, и необходимо добавить в таблицу ещё k элементов с такими же значениями хеш-функции: множество R будет множеством ячеек, которые займут добавляемые элементы.

Также отметим, что процесс дешифрования полностью симметричен процессу шифрования, только необходимо вычитать число 1, а не прибавлять. Для дешифрования можно заменить каждое значение t_i на $n + 1 - t_i$, развернуть последовательность чисел, применить тот же алгоритм, что и для шифрования, а затем сделать такое же преобразование ещё раз.

Задача 3. Рекорды и антирекорды

Автор:	Федор Ромашов
Разработчик:	Мария Жогова

Подзадача 1.

В данной подзадаче малые ограничения на n позволяют перебрать всевозможные разбиения p на q и r. Для каждого из них за $O(n)$ линейным проходом насчитывается число рекордов и антирекордов в соответствующих перестановках. Таким образом решение работает за $O(2^n \cdot n)$ на один тестовый случай.

Подзадача 2.

Воспользуемся динамическим программированием для решения данной задачи. Введём $dp_{i,a,b}$ — максимальный ответ, если разбить префикс длины i, так чтобы последний рекорд в q был равен a, а последний антирекорд в r был равен b.
Тогда, существуют следующие переходы из $dp_{i,a,b}$:

- $dp_{i+1,a,b} \leftarrow dp_{i,a,b}$, если $p_{i+1} < a$, берем в q
- $dp_{i+1,a,b} \leftarrow dp_{i,a,b}$, если $p_{i+1} > b$, берем в r
- $dp_{i+1,p_{i+1},b} \leftarrow dp_{i,a,b} + 1$, если $p_{i+1} > a$, берем в q
- $dp_{i+1,a,p_{i+1}} \leftarrow dp_{i,a,b} + 1$, если $p_{i+1} < b$, берем в r

Всего $O(n^3)$ состояний, из каждого из которых $O(1)$ переходов. Таким образом решение работает за $O(n^3)$ на один тестовый случай.

Подзадачи 3-4.

Рассмотрим динамику из предыдущей подзадачи.
Разделим состояния на два типа:

1. Состояние $dp_{i,a,b}$, где $a > b$. Заметим, тогда для любого x, либо $a > x$, либо $b < x$. Тогда взяв элемент x в соответствующую подпоследовательность, можем не изменить рекорд и антирекорд. Таким образом, из оставшегося суффикса можно выбросить любой элемент, не изменив ответ. Тогда выберем на суффиксе длины $n - i$ какую-то подпоследовательность p' в качестве рекордов, больших a, и подпоследовательность q' антирекордов, меньших b, выкинув элементы, не попавшие в p' или q'. Для максимизации ответа необходимо выбрать в качестве p' НВП (наибольшую возрастающую последовательность), а в качестве q' НУП (наибольшую убывающую последовательность) на суффиксе длины $n - i$.

2. Состояние $dp_{i,a,b}$, где $a < b$. Заметим, что в таком состоянии, мы разбили префикс длины i на p и q так, что максимальный элемент в p равен a, а минимальный элемент в q равен b. Таким образом, все элементы $\leqslant a$ находятся в p, а все элементы $\geqslant b$ находятся в q. Таких разбиений префикса длины i не более чем $i + 1$. Суммарно получаем, что состояний типа *2* — $O(n^2)$.

Насчитаем для каждого i, x : $LIS(i, x)$ — НВП на суффиксе длины $n - i$, начинающуюся с элемента $> x$. Аналогично, насчитаем $LDS(i, x)$ — НУП, начинающуюся с элемента $< x$. Эта часть работает за $O(n^2 \log n)$ или $O(n^2)$ в зависимости от реализации.

Далее поддерживаем динамику только для состояний типа *2* и при переходе в состояние типа *1* обновляем ответ за $O(1)$.

Для подзадачи 3 — реализация за $O(n^2 \log n)$, для 4 — за $O(n^2)$.

Подзадача 6.

Заметим, что динамику из подзадач 2–4 можно пересчитывать эффективнее, используя дерево отрезков.

А именно насчитывать НУП и НВП можно с помощью дерева отрезков по увеличению длины суффикса. Однако, так как в динамике нам нужно увеличивать длину префикса, то НУП и НВП можно откатывать. Эта часть работает за $O(n \log n)$.

Далее, чтобы все состояния типа *2* —(i, a, b) можно поддерживать в одном из своих элементов к примеру в a в дереве отрезков. При рассмотрении элемента $p_i = x$:

1. Удаляется ровно 1 состояние типа *2* (i, a, b), такое что $a < x < b$ и добавляются два новых состояния типа *2* — $(i + 1, a, x)$ и $(i + 1, x, b)$.

2. Из всех состояний типа *2* (i, a, b), таких что $x < a < b$, можно сделать переход в состояние типа *1* — $(i + 1, a, x)$. При этом из него ответ будет как $dp_{i,a,b} + 1 + LIS(i + 1, a) + LDS(i + 1, x)$.

3. Аналогично 2, но $a < b < x$.

Таким образом, если поддерживать в одном дереве отрезков в элементе a значение $dp_{i,a,b} + LIS(i + 1, a)$, обновлять ответ для второго случая можно за $O(\log n)$. Аналогично обрабатывается случай 3.

Суммарно получаем решение за $O(n \log n)$ на один тестовый случай.

Задача 4. Ультра mex

Автор:	Владимир Новиков
Разработчики:	Александр Голованов, Иван Сафонов

При $k \leqslant 4$ можно решить задачу перебором всех возможных множеств A ($O(2^{2^k})$ штук) и симуляцией процесса для них.

Рассмотрим то, как работает наш процесс. Рассмотрим битовый бор двоичных записей всех чисел множества (каждое число рассматривается от старшего бита к младшему). Глубина такого бора равна k.

Заметим, что если наш бор полный, то есть наше множество состоит из всех чисел и имеет размер 2^k, то mex предел также равен 2^k.

Пусть наш бор не полный. Рассмотрим левое и правое поддеревья корня нашего бора.

- Если левое поддерево заполнено не полностью, то $mex = m$ будет $< 2^{k-1}$, следовательно мы сделаем операцию $\oplus(m - 1)$. После такой операции все элементы которые были в левом поддереве бора останутся там, все которые были в правом поддереве бора тоже останутся там. При этом в левом поддереве останется хотя бы один отсутствующий элемент. Значит в правом

поддереве нашего бора может быть произвольное подмножество, мы можем рассматривать только левое поддерево и убрать из всех чисел старший бит, mex-предел от этого не изменится.

- Пусть все левое поддерево бора полностью заполнено. Рассмотрим правое поддерево. Допустим, что минимальный элемент 2^{k-1} правого поддерева есть в множестве. Тогда $mex = m > 2^{k-1}$ следовательно мы сделаем операцию $\oplus(m-1)$ с числом $m-1 \geqslant 2^{k-1}$. Тогда заметим, что после операции поддеревья поменяются местами и мы получим первый случай. Но тогда поменяем поддеревья до выполнения операции. Заметим, что такой бор будет удовлетворять первому случаю и mex-предел множества не изменится после замены.

- Пусть все левое поддерево бора полностью заполнено. Рассмотрим правое поддерево. Допустим, что минимального элемента 2^{k-1} нет в множестве. Тогда $mex = 2^{k-1}$ и мы делаем операцию $\oplus(2^{k-1}-1)$. Такая операция не меняет поддеревья бора местами, а переворачивает их. Заметим, что если в изначальном множестве не было числа 2^k-1, то теперь снова левое поддерево заполнено, а правое не содержит 2^{k-1}. Процесс зациклился и теперь все получаемые mex будут равны 2^{k-1}. Мы получили что mex-предел нашего множества равен 2^{k-1}.

- Остался последний случай, когда левое поддерево бора полностью заполнено, в правом поддереве нет числа 2^{k-1}, но есть число 2^k-1. В таком случае мы сначала сделаем операцию $\oplus(2^{k-1}-1)$, перевернув поддеревья, затем мы получим такую же ситуацию как в случае 2. То есть mex-предел нашего множества такой же как mex-предел перевернутого правого поддерева.

Таким образом, в случаях 1, 2, 4 мы получаем бор глубины $k-1$ с таким же mex-пределом, в случае 3 мы получаем, что mex-предел множества равен 2^{k-1}. Таким образом, mex-предел может быть равен только степени 2 и если p не степень 2, то ответ равен 0. Далее будем считать что $p = 2^q$.

Зафиксируем q. Напишем динамическое программирование $dp[k][n][last]$, где $1 \leqslant n \leqslant 2^k$ и $last \in \{0, 1\}$. Здесь:

- $dp[k][n][0]$ равно количеству подмножеств чисел от 0 до 2^k-1 содержащих 0, таких что их размер равен n и mex-предел равен q.

- $dp[k][n][1]$ равно количеству подмножеств чисел от 0 до 2^k-2 содержащих 0, таких что их размер равен n и mex-предел равен q. В этом случае элемент 2^k-1 не должен лежать в подмножестве.

Инициализация (соответствует случаю 3):

- $dp[q][2^q][0] = 1$, все остальные значения с $k \leqslant q$ равны 0

- $dp[q+1][2^q+n][0] = dp[q+1][2^q+n][1] = C^n_{\max(2^q-2,0)}$, все остальные значения при $k > q+1$ равны 0

Формулы вычисления (при $k \geqslant q+2$):

- $dp[k][n][0] += \sum_{r=0}^{2^{k-1}} dp[k-1][n-r][0] \cdot C^r_{2^{k-1}}$; $dp[k][n][1] += \sum_{r=0}^{2^{k-1}-1} dp[k-1][n-r][0] \cdot C^r_{2^{k-1}-1}$
  Эти переходы соответствуют случаю 1, когда левое поддерево является не полным.

- $dp[k][2^{k-1}+n][0] += dp[k-1][n][0] + dp[k-1][n][1]$ (при $n \geqslant 1$)
  Первое слагаемое соответствует случаю 2, второе слагаемое соответствует случаю 4

- $dp[k][2^{k-1}+n][1] += dp[k-1][n][1]$ (при $n \geqslant 1$)
  Единственное слагаемое соответствует случаю 2, случай 4 при $last = 1$ невозможен, так как по определению элемента 2^k-1 при $last = 1$ в множестве быть не должно

Ответ на задачу для тройки $(k, n, 2^q)$ будет лежать в значении $dp[k][n][0]$.

Таким образом, напрямую по формулам представленным выше можно вычислить все ответы при фиксированном q за время $O(2^{2k})$. То есть общая асимптотика $O(k4^k)$, что может пройти подзадачи для $k \leqslant 12$.

Задачу можно решить эффективнее для одной тройки. Для этого зафиксируем путь от поддерева размера 2^q до корня бора. Всего их 2^{k-q} штук. Заметим, что все поддеревья висящие на пути влево должны быть полными. Все поддеревья висящие на пути справа могут содержать произвольные подмножества, за исключением случаев того, содержатся ли элементы 2^{i-1}, $2^i - 1$. Тогда можно разделить оставшиеся элементы, которые не попали в левые поддеревья между правыми поддеревьями, висящими на пути, поэтому такое значение равно одному биномиальному коэффициенту. Такое решение работает за $O(k2^{k-q})$ на один тест, что позволяет пройти подгруппы с $t \leqslant 10$ и произвольным k.

Для полного решения можно ускорить вычисление изначального динамического программирования с помощью быстрого преобразования фурье. Мы можем заметить, что переходы первого вида при вычислении работают за $O(2^{2k})$, если их вычислять наивно. Заметим, что то что нужно вычислить это произведение двух многочленов $P_i = dp[k-1][i][0]$ и $Q_i = C^i_{2^{k-1}}$. Произведение можно вычислить за $O(k2^k)$. Тогда суммарное время работы при фиксированном q будет $\sum\limits_{i=q+2}^{k} i2^i = O(k2^k)$, поэтому общая асимптотика $O(k^2 2^k)$.