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 $O(k^2 2^k)$.

Задача 5. Улитка на склоне

Автор:	Денис Кириенко
Разработчик:	Тихон Евтеев

Посетим все вершины дерева с помощью рекурсивного обхода в глубину, подсчитывая количество поворотов, встречающихся на пути к каждой вершине.

Для этого будем передавать в рекурсивную функцию обхода, помимо самой вершины, также количество уже пройденных поворотов и направление спуска, выбранное в предыдущей вершине. Начнём рекурсивный обход с корня дерева (вершины номер 1), приняв за предыдущее направление спуска специальное значение «спуск отсутствует».

Также функция рекурсивного обхода в глубину будет для каждой вершины возвращать количество листьев в её поддереве, в которых может заканчиваться подходящий маршрут. Если вершина является листом, то функция возвращает 0 или 1, в зависимости от числа совершённых поворотов, а для других вершин функция возвращает сумму значений функций для её потомков. Это значение будем сразу же сохранять в массиве, что позволит отвечать на каждый запрос после этого за $O(1)$. Таким образом, сложность решения будет $O(n + q)$.

Задача 6. Конференция

Автор:	Владимир Новиков
Разработчик:	Максим Деб Натх

Для решения этой задачи важной подзадачей будет определение размера максимального совместного множества для произвольного набора отрезков. Для решения этой подзадачи можно воспользоваться жадным алгоритмом. Сначала надо отсортировать все события по увеличению r_i. После этого надо перебрать отрезки в порядке сортировки и брать очередной отрезок в ответ только, если оно не пересекается с предыдущим. Несложно показать, что данный жадный алгоритм всегда находит корректный ответ к подзадаче.

Группа 1. никакие два отрезка не пересекаются

Для решения подзадачи достаточно заметить, что само ограничение говорит о том, что все отрезки лежат в максимальном по размеру множестве непересекающихся отрезков, поэтому удалив любые $\frac{n}{2}$ из них, получим корректный ответ под условия задачи.

Группа 2. $n \leqslant 20$**:** $\mathcal{O}(n \log n 2^n)$

Для решения переборной подзадачи достаточно перебрать множество отрезков, которое останется после удаления (а именно, перебрать все подмаски исходного множества). Для проверки ответа на оставшемся множестве нужно воспользоваться описанной выше проверкой за $\mathcal{O}(n \log n)$.

Группа 3. $n \leqslant 30$**:** $\mathcal{O}\left(nC_n^{n/2}\right)$

Перебор из предыдущей подгруппы можно оптимизировать: во-первых, заметив, что мы каждый раз сортируем все отрезки по правой границе, можно отсортировать их только один раз, избавившись от логарифмического множителя в асимптотике.

Кроме этого, можно оптимизировать перебор, переписав его как перебор с возвращением: напишем рекурсивную функцию, которая будет перебирать все подмножества и параллельно с этим поддерживать размер максимального совместного множества. Чтобы перебор был эффективным, будем выходить из рекурсии, если дальнейший перебор не даст нам корректного подмножества.

Такое решение переберёт все валидные $C_n^{n/2}$ подмасок и найдёт ответ быстрее, чем предыдущее.

Группа 4: отрезки образуют древовидную структуру, $n \leqslant 500$

Ограничение «В каждой паре мероприятий либо одно мероприятие накрывает другое, либо они не пересекаются, существует мероприятие, которое накрывает все остальные» говорит о том, что отрезки можно изобразить как дерево вложенности. В таком дереве для каждого отрезка, кроме корневого, можно выделить его непосредственного предка – отрезок, в котором он лежит.

Нетрудно видеть, что в таком дереве максимальным совместным множеством будет количество листьев.

Заметим, что если мы получили какое-то множество размера n, у которого размер максимального совместного множества равен $k < n$, то можно выкинуть какой-то отрезок из $n - k$ не лежащих в максимальном совместном множестве отрезков и мы всё равно получим множество с ответом k.

Тогда воспользуемся динамическим программированием по поддеревьям и вычислим для каждой вершины u следующую величину: `dp[u][l]` — максимальный размер множества в поддереве вершины u, для которого ответ равен l.

Группа 7: $n \leqslant 5000$**:** $\mathcal{O}(n^2)$

Для решения подзадачи с $n \leqslant 5000$ воспользуемся следующей идеей: найдем множество s_1 , на котором размер максимального совместного множества $\leqslant \frac{ans}{2}$ и множество s_2, на котором этот размер $\geqslant \frac{ans}{2}$, где ans — размер максимального совместного множества на исходном множестве отрезков.

Будем постепенно преобразовывать множество из s_1 в s_2.

Для этого мы будем брать произвольный элемент t_1 из s_1, который не лежит в s_2 и заменять его на элемент t_2 из s_2, который не лежит в s_1. Заметим, что при удалении t_1 из s_1 ответ может только уменьшиться, а после вставки в s_1 элемента t_2 ответ может только увеличиться. Нетрудно заметить, что при переходе от s_1 к $s_1 - t_1 + t_2$ ответ поменяется не более, чем на 1 (в любую из сторон). Таким образом, в какой-то момент выполнения преобразований мы найдем множество, на котором ответ будет ровно $\frac{ans}{2}$.

Реализация данного алгоритма без оптимизаций дает нам асимптотику $O(n^2 logn)$, так как для каждого из n множеств мы будем запускать жадный алгоритм. Данное решение при хорошей реализации проходит подгруппу, но чтобы добиться чистой оценки $O(n^2)$, достаточно в жадном алгоритме не запускать сортировку каждый раз, а предподсчитать порядок сортировки отрезков.

Полное решение: $\mathcal{O}(n \log n)$

Для полного решения сделаем следующим образом: найдем первые $\frac{ans}{2}$ отрезков с помощью жадного алгоритма при сортировке по правому концу и проходу слева направо, а также первые $\frac{ans}{2}$ отрезков при сортировке по левому концу и проходу справа налево. Назовём эти множества L и R: $|L| = |R| = \frac{ans}{2}$.

Тогда все остальные отрезки можно разбить на две группы: те, которое пересекаются с каким-то отрезком из L, и все остальные. Назовём эти группы L' и R'. Заметим, что $|L| + |L'| + |R| + |R'| = n$. А значит, что либо $|L| + |L'|$, либо $|R| + |R|'$ будет не меньше, чем $\frac{n}{2}$. То есть, можно взять большее из множеств $L \cup L'$ и $R \cup R'$, ответ в нём будет искомым, но само множество может превосходить требуемый размер в $\frac{n}{2}$. В таком случае просто выкинем из него какие-то отрезки, которые не лежат в ответе, чтобы получить требуемый размер.

Задача 7. Яблоки по корзинам

Автор: Александр Бабин
Разработчики: Иван Сафонов, Тихон Евтеев, Николай Будин

Подзадача 1

В этой подзадаче для нахождения ответа на запрос, можно было использовать перебор всех возможных вариантов распределить яблоки по двум корзинам (и часть оставить на столе). Тогда нужно будет проверить, что все пары (x, y) $(0 \leqslant x \leqslant a,\ 0 \leqslant y \leqslant b)$ являются достижимыми. Такое решение работает за $O(3^n \cdot n \cdot q)$.

$z = 0$

Во всех подзадачах, кроме последней, гарантируется, что $z = 0$. Это означает, что запросы даны в незакодированном виде. И можно отвечать на них в «оффлайне», то есть сначала считать все запросы, при необходимости переупорядочить, вычислить ответы, и в конце вывести все ответы. Мы будем пользоваться этим для того, чтобы вычислять ответы на запросы в порядке возрастания k.

$k = 10^{18}$

В подзадачах 2–9 гарантируется, что во всех запросах $k = 10^{18}$. Это означает, что в каждом запросе мы рассматриваем полное множество яблок, данное в тесте. Поэтому, можно сначала обработать все яблоки, вычислить какую-то информацию для полного множества яблок, и потом отвечать на запросы.

Произвольные k

В подзадачах 10–17 k в запросах могут быть произвольными, но все еще верно, что $z = 0$. Поэтому, мы считаем все запросы, переупорядочим их по возрастанию k. Затем, возьмем пустое множество и будем добавлять туда яблоки в порядке возрастания веса. И перед тем, как добавить новое яблоко, вычислим ответы на все запросы, в которых k меньше, чем вес добавляемого яблока, и на которые мы не ответили раньше.

$b = 0$

Случай $b = 0$ является одномерной задачей. В одномерной задаче можно использовать достаточно простую жадность. Будем поддерживать минимальную сумму, которую нельзя собрать. Для пустого множества это 1. Затем можно добавлять яблоки в порядке возрастания веса и пересчитывать эту величину.

Рюкзак

Подзадачи 4–6 и 11–14 рассчитаны на решения, использующие метод динамического программирования. А именно, реализующие двухмерный «рюкзак». В том числе, с битовым сжатием для ускорения.

Полное решение

Одним из подходов к решению является анализ минимальных по включению неидеальных пар. Если мы научимся поддерживать такое множество B, то для ответа на запрос, нужно будет проверить, есть ли в B элемент (x, y): $x \leqslant a$ и $y \leqslant b$.

Возьмем пустое множество яблок, и будем добавлять туда яблоки в порядке возрастания веса. До какого-то момента достижимыми будут являться все пары (x, y): $x + y \leqslant \sum_{j=1}^{i} w_j$. А именно, пока

для всех яблок на префиксе будет верно, что $w_i \leqslant \lfloor \frac{\sum_{j=1}^{i-1} w_j}{2} \rfloor + 1$. Значит, B это $(0, S+1)$, $(1, S)$, $\ldots$, $(x, S+1-x)$, $\ldots$, $(S+1, 0)$, где S — сумма весов добавленных яблок.

Затем, когда мы впервые встретим яблоко, для которого такой критерий не выполнен, множество B начнет изменяться следующим образом. Все пары (x, y) из B, для которых верно $w_i \leqslant x$, превратятся в $(x + w_i, y)$. Все пары, для которых верно $w_i \leqslant y$, превратятся в $(x, y + w_i)$. Причем, каждая точка будет относиться максимум к одному из этих двух случаев.

Если поддерживать B как множество точек в массиве, то можно сделать решение за $O(n \cdot C + q \cdot \log C)$, где C — ограничение на a и b.

Если поддерживать B как множество точек, но в какой-нибудь структуре данных, то можно сделать решение за $O(n \cdot \log C + q \cdot \log C)$.

Чтобы решить задачу при максимальных ограничениях на C, нужно хранить B как набор диагональных отрезков. Этого уже достаточно для полного решения, но дополнительно можно заметить, что количество таких отрезков в B не будет превышать $\log C$ в силу особенностей перестроения.

Произвольное z

Чтобы решить задачу при произвольном z, нужно использовать решение для $z = 0$, но сначала обработать все яблоки и запомнить все промежуточные состояния структуры данных, а потом отвечать на запросы, обращаясь к нужным версиям структуры данных. Для этого, можно либо сделать структуру данных персистентной, либо воспользоваться тем, что если хранить B как набор отрезков, его размер будет $O(\log C)$, и просто сохранить все n множеств.

Альтернативная идея полного решения

Утверждается, что пара (a, b) является идеальной тогда и только тогда, когда для всех (x, y): $0 \leqslant x \leqslant a$ и $0 \leqslant y \leqslant b$ выполнено следующее. Сумма весов яблок с весами не превышающими $\min(k, \max(x, y))$ не меньше, чем $x + y$.

На основе этой идеи также можно реализовать полное решение.

Задача 8. Выполнить план, но не перевыполнить

Автор:	Александр Бабин
Разработчик:	Иван Сафонов

Заметим, что в задаче эффективностью плана называется размер максимального по весу независимого множества в дереве, где вес i-й вершины это a_i.

В подзадаче 1 выполнено, что $l_i = r_i$. В этом случае существует единственный план, найдем его эффективность. Чтобы посчитать эффективность, можно использовать динамическое программирование по поддеревьям. Обозначим $dp[p][0]$ как максимальный вес независимого множества в поддереве вершины p, такого что p в нем точно не лежит; $dp[p][1]$ как максимальный вес независимого множества в поддереве вершины p, такого что p может в нем лежать, а может не лежать. Тогда делая dfs из корня можно посчитать это динамическое программирование. Если мы посчитали его в сыне q вершины p, то надо сделать прибавления $dp[p][0] += dp[q][1]$, $dp[p][1] += dp[q][0]$. В конце надо сделать $dp[p][1] = \max(dp[p][1], dp[p][0])$. Изначальные значения $dp[p][0] = 0$, $dp[p][1] = a[p]$.

В подзадаче 2 выполнено, что $c = 0$. Это означает, что никакой план проверяться не будет. Фактически это Yes/No задача проверки существования плана, потому что в случае существования сертификаты проверяться не будут. Давайте найдем число L — это максимальный вес независимого множества, если веса вершин равны l_i, R — это максимальный вес независимого множества, если веса вершин равны r_i. Заметим, что если $v_i < L$ или $v_i > R$, то плана точно не существует. Утверждается, что если $v_i \in [L, R]$, то план с эффективностью v_i всегда существует. Позже мы это покажем.

В подзадаче 3 выполнено, что $l_i = 0$, $r_i \leqslant 1$. Веса вершин могут быть от 0 до 1. Найдем максимальное по весу независимое множество I. Можно там оставить только вершины с $r_i = 1$. Тогда

заметим, что существуют планы с эффективностями $v_i \in [0, |I|]$. Чтобы построить план с эффективностью v_i, можно выставить первым v_i элементам из I вес 1, всем остальным вес 0. Чтобы найти сертификаты, можно воспользоваться префиксными ксорами.

Несложно понять, что это решение обобщается на подзадачу 4. В этом случае также можно найти максимальное по весу независимое множество I и существуют планы с эффективностями $v_i \in [0, \sum\limits_{i \in I} r_i]$. Чтобы построить план с эффективностью v_i, можно взять префикс I с суммой v_i. Веса элементов префикса будут равны v_i, вес последнего элемента префикса может получиться произвольным, веса остальных элементов равны 0. Сертификаты таких планов можно посчитать также с помощью префиксных ксоров быстро.

В подзадаче 6 можно перебрать все возможные планы за $O(\prod\limits_{i=1}^{n}(r_i - l_i + 1))$ и для каждого посчитать эффективность, запомнив все ответы.

Рассмотрим подзадачу 8. Там есть дополнительное условие $\sum\limits_{i=1}^{n} r_i - l_i \leqslant 10^4$. Докажем наш критерий существования плана и покажем, как его строить. Пусть у нас есть некоторый план $[a_1, a_2, \ldots, a_n]$, который имеет эффективность v. Тогда увеличим одно из a_i на 1. Заметим, что эффективность нового плана $v' \in [v, v+1]$. То есть она либо не изменится, либо увеличится на 1. Иногда такое свойство называют дискретной непрерывностью. В данном случае это свойство нам очень помогает: давайте начнем с плана $[l_1, l_2, \ldots, l_n]$ с эффективностью L и увеличивая на 1 элементы плана дойдем до плана $[r_1, r_2, \ldots, r_n]$ с эффективностью R. Заметим, что в ходе этого процесса мы получим все возможные эффективности из отрезка $[L, R]$. Получаем решение за $O(n(\sum\limits_{i=1}^{n}(r_i - l_i + 1)))$.

Заметим, что для того, чтобы найти план с эффективностью v_i мы можем воспользоваться бинарным поиском, вместо линейного перебора всех планов. Пусть $f(x)$ это эффективность плана с суммой x, который строится так: сначала на некотором префиксе $a_i = r_i$, затем одно значение a_i произвольное, затем $a_i = l_i$. Функция f не убывающая и мы знаем, что на концах отрезка сумм она равна L и R. Тогда чтобы найти x, такое что $f(x) = v_i$ можно сделать бинарный поиск. Такое решение работает за $O(nq \log C)$, где $C = 10^9$.

Рассмотрим подробно план, в котором $a_j = r_j$ при $j < i$, $a_i = x \in [l_i, r_i]$ произвольное число и $a_j = l_j$ при $j > i$. Подвесим дерево за вершину i. Тогда заметим, что веса всех вершин кроме корня зафиксированы, значит мы можем посчитать наше динамическое программирование для поиска максимального по весу независимого множества из нашего корня. Тогда заметим, что вес максимального независимого множества во всем дереве равен $\max(A, B + x)$, где $A = \sum\limits_{(i,j) \in E} dp[j][1]$, $B = \sum\limits_{(i,j) \in E} dp[j][0]$ (E это множество ребер). Тогда мы можем ответить на все запросы, такие что $v_i \in [\max(A, B + l_i), \max(A, B + r_i)]$. Такое решение работает за $O(n^2 + q \log n)$.

Рассмотрим полное решение. Давайте будем рассматривать прошлое решение, но будем идти по всем вершинам не в порядке $[1, 2, \ldots, n]$, а в порядке эйлерова обхода нашего дерева из корня 1. Заметим, что тогда динамические программирования, которые нам нужно вычислить при подвешивании за i это динамические программирования в дереве подвешенном за 1 и весами вершин l_i или весами вершин r_i. Также одно поддерево у нас получается по ребру вверх из вершины i, но все веса вершин, которые мы уже обошли равны r_i, всех вершин которые не обошли l_i. Заметим, что динамическое программирование для этой части дерева также можно поддерживать в ходе обхода dfs. Таким образом функцию $\max(A, B + x)$ мы получаем не с помощью вычисления динамического программирования с нуля, а переиспользуя динамическое программирование из корня 1 и всеми весами либо l_i, либо r_i. Такое решение работает за $O(n + q \log n)$.

Для эффективного вычисления сертификатов в подзадачах с $q > c$ можно использовать префиксные ксоры, это небольшая техническая деталь решения.