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

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.