Задача A. Колония бактерий

Автор задачи: Егор Юлин, разработчик: Николай Ведерников

Задача на вывод формулы. Есть много закономерностей, которые нужно заметить для вывода итоговой формулы. Рассмотрим одну из них.

Рассмотрим горизонталь и вертикаль, в которую изначально помещается бактерия. Каждую секунду она увеличивается вверх и вниз, следовательно, в секунду k колония будет помещаться в квадрат $(2k-1) \times (2k-1)$. Но при этом каждую нечётную секунду у нас растёт не заполненный `уголок` квадрата. Можно заметить, что его длина будет равна $\lfloor (k-1)/2 \rfloor$. А следовательно, площадь будет равна $1+2+..+\lfloor (k-1)/2 \rfloor$. По формуле арифметической прогрессии получаем, что площадь уголка равна $\lfloor (k-1)/2 \rfloor \cdot \lfloor (k+1)/2 \rfloor /2$. Всего таких уголков четыре, значит, из квадрата у нас отрежется $4 \cdot \lfloor (k-1)/2 \rfloor \cdot \lfloor (k+1)/2 \rfloor /2 = 2 \cdot \lfloor (k-1)/2 \rfloor \cdot \lfloor (k+1)/2 \rfloor /2$ клеток. Итого площадь колонии бактерий будет равна $(2k-1) \cdot (2k-1) - 2 \cdot \lfloor (k-1)/2 \rfloor \cdot \lfloor (k+1)/2 \rfloor /2$.

Задача B. Двухэтажный адвент-календарь

Автор и разработчик задачи: Рита Саблина

Рассмотрим одно из возможных решений этой задачи.

Сначала мы представим каждую коробку как отрезок на линии. Каждая коробка i характеризуется своими левым и правым границами l_i и r_i, а также числом x_i, написанным на ней. Первые коробки на обоих уровнях имеют левую границу в точке 0.

Мы разделим все коробки на два типа: хорошие и проблемные. Для хорошей коробки, если она находится на верхнем уровне, то все коробки, на которых она лежит, имеют большее число, и наоборот, для хорошей коробки на нижнем уровне. Проблемные коробки находятся под коробками с большим числом или лежат на коробках с меньшим числом. В этой задаче коробки, которые пересекаются только на своих концах, не мешают друг другу. Мы не будем считать коробки, которые пересекаются только на своих концах, лежащими друг на друге.

Формально, коробка i с границами l_i и r_i и числом x_i считается хорошей, если для любой коробки j на другом уровне с границами и числом l_j, r_j и x_j, если $l_j \leqslant l_i$ и $r_j > l_i$, или $l_j < r_i$ и $r_j \geqslant r_i$, или $l_i \leqslant l_j < r_j \leqslant r_i$, то $x_i < x_j$, если i находится на верхнем уровне, и $x_i > x_j$ в противоположном случае. Проблемные коробки — это все остальные.

Обратите внимание, что хорошую коробку всегда можно оставить на месте. Мы определим для каждой коробки, используя два указателя, является ли она хорошей или проблемной. Также для каждой проблемной коробки i на верхнем уровне мы найдем правую границу самой правой проблемной коробки j на нижнем уровне, на которой она лежит, а для каждой проблемной коробки j на нижнем уровне мы найдем правую границу самой правой коробки i на верхнем уровне, под которой она лежит. Кроме того, для каждой проблемной коробки i на верхнем уровне нам нужно будет сохранить правый конец коробки j на нижнем уровне так, чтобы $l_j < l_i < r_j$, если такая коробка существует. Для каждой проблемной коробки j на нижнем уровне мы будем сохранять правый конец коробки i на верхнем уровне так, чтобы $l_i < l_j < r_i$, если такая коробка существует. Мы будем называть эту границу r. Если такая коробка не может быть найдена, мы сохраним левый конец текущей коробки в r.

Далее мы решим эту задачу с помощью динамического программирования. Параметром динамического программирования будут координаты концов коробок. Мы будем представлять координаты как события и сортировать их в неубывающем порядке. Мы обозначим $dp[i]$ как минимальное количество коробок, которые необходимо удалить, чтобы сделать календарь от координаты 0 до i включительно удобным. Обратите внимание, что в массиве событий могут быть пары одинаковых координат, если коробки заканчиваются в одной и той же точке. Мы устанавливаем $dp[0] = 0$.

Если коробка хорошая, то $dp[i] = dp[i-1]$.

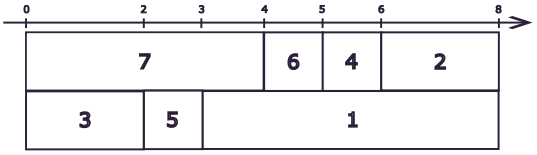
Если коробка с концом в i проблемная и она лежит на или под проблемной коробкой, которая заканчивается справа от нее, то коробку i нельзя оставить, если мы хотим сделать календарь от 0 до i удобным. В этом случае $dp[i] = \min(dp[l]+1, dp[r]+1)$, где $dp[l]$ — минимальное количество коробок, которые нужно было удалить перед левой границей текущей коробки. r может совпадать

с l, но если нет, то $dp[r]$ может быть меньше, чем $dp[l]$ — если результат в конце коробки, на которой лежит текущая, лучше, но он был рассчитан позже, так как текущая коробка начинается до того, как заканчивается та, на которой она лежит или под которой.

Если коробка с концом в i проблемная и она лежит на или под проблемной коробкой, которая заканчивается слева от нее или в той же точке, то нам нужно выбрать лучший результат между удалением текущей коробки $dp[i] = \min(dp[l] + 1, dp[r] + 1)$ и $dp[i-1]$ — предыдущая координата будет соответствовать ситуации, когда мы оставляем текущую, но удаляем проблемную, на или под которой лежит текущая.

Ответ будет в $dp[a_1 + a_2 + \ldots + a_n]$.

Пример на рисунке:



$dp[0] = 0$. $dp[2] = 1$, потому что на коробочке с номером 3 лежит коробочка 7, которая заканчивается правее, поэтому 3 не убрать нельзя, чтобы к координате 2 календарь стал удобным. $dp[3] = 1$, потому что на коробочке с номером 5 лежит коробочка 7, которая заканчивается правее, поэтому 5 не убрать нельзя, чтобы к координате 3 календарь стал удобным. Тогда $dp[3] = dp[2] + 1$, где 2 — левая граница коробочки 5.

$dp[4] = dp[0] + 1 = 1$, потому что не убрать коробочку 7 нельзя, чтобы к координате 4 календарь стал удобным. $dp[5] = dp[4] + 1 = 2$, $dp[6] = dp[5] + 1 = 3$ по такой же причине. $dp[8]$ будет рассматриваться два раза. Если рассматривать 8 как правую границу коробочки 2, то не убрать ее нельзя, и $dp[8] = dp[6] + 1 = 4$. Когда мы будем рассматривать 8 как правую границу коробочки 1, то выгодно убрать коробочку 1, а не 2, 4, 6, под которыми она лежит, но самая левая проблемная коробочка, которая лежит на 1, — 7 — заканчивается правее, чем начинается 1, и значение динамики для правого конца 7 лучше. $dp[8] = min(dp[3], dp[4]) + 1 = 2$. В итоге ответ для этого примера будет 2 — нужно убрать коробочки 1 и 1.

Задача C. Промежуточная вертикальность

Автор и разработчик задачи: Михаил Иванов

Оказывается, в любом связном графе можно за линейное время построить дерево любой промежуточной вертикальности. Начнём с дерева с максимальной возможной вертикальностью — дерева поиска в глубину, имеющего вертикальность $m - n + 1$. Если мы $(m - n + 1) - h$ раз сделаем преобразование, увеличивающее число горизонтальных рёбер на единицу, то получим требуемое дерево. Для начала научимся делать это преобразование за линейное время. Обойдём граф вдоль остовного дерева и найдём вертикальное ребро uv с самой большой глубиной (расстоянием до корня по древесным рёбрам) у вершины u (считаем, что глубина v больше глубины u). Так как у v положительная глубина, эта вершина имеет родителя w; таким образом, в графе есть древесное ребро wv. Тогда давайте из остовного дерева выкинем ребро wv и вместо него добавим uv. Тогда ни у какого ребра, кроме uv и wv, не поменяется статус — это легко установить разбором случаев, пользуясь как раз тем, что u самая глубокая; wv из древесного станет горизонтальным, а uv станет из вертикального древесным. Таким образом, требуемое преобразование реализуемо за линейное время, что даёт квадратичный алгоритм со временем работы $\mathcal{O}(m(m + 1 - n - h))$.

Чтобы всё то же самое проделать суммарно за линейное время, требуется лишь научиться делать описанное выше преобразование за $\mathcal{O}(1)$ времени (возможно, с линейным предподсчётом). Заметим, что требуемое переподвешивание не только не добавляет новых вертикальных рёбер (и убирает всегда только одно старое, а остальные не меняет), но и не может поменять глубину верхней вершины u никакого вертикального ребра uv. Следовательно, сработает такой алгоритм: запустим один раз в самом начале поиск в глубину и получим нормальное остовное дерево. В процессе сохраним у каждой вершины родителя в массиве p, а также с помощью сортировки подсчётом отсортируем все вертикальные рёбра по глубине их верхней вершины. Переберём $(m - n + 1) - h$ последних рёбер

из этого списка (то есть с самой глубокой u) и у них присвоим $p[v] := u$. Горизонтальными станут $(m - n + 1) - h$ рёбер, соединяющих v и $p[v]$. У этого алгоритма асимптотика уже $\mathcal{O}(m)$.

Задача D. Два массива

Автор и разработчик задачи: Павел Скобелин

Давайте переформулируем задачу. Давайте воспринимать пару чисел (a_i, b_i) как точку на плоскости. Тогда с помощью заданного действия мы умеем двигать эту точку по диагонали на одну клетку $(+1, +1)$. Тогда с помощью данного действия нужно поместить все точки в прямоугольник размера $x \times y$.

Понятно, что конкретная точка всегда будет оставаться на одной и той же диагонали. Пронумеруем диагонали: скажем, что точка (x, y) принадлежит диагонали с номером $y-x$ (номер диагонали может быть отрицательным).

Тогда критерий невозможности ответа понятен: найдем номер максимальной диагонали — D_{max}, и минимальной — D_{min}, на которых есть хотя бы одна точка. Тогда несложно показать, что ответ всегда существует, если $D_{max} - D_{min} \leqslant x + y$ (ведь в прямоугольнике размера $x \times y$ максимальная разность номеров диагоналей это $x+y$ — то есть левой верхней и правой нижней точки). В противном случае ответа не существует.

Теперь поймем, как искать количество необходимых действий. Положим:

$$X = \max_{i \in [1...n]}(x_i), Y = \max_{i \in [1...n]}(y_i)$$

Наша цель — загнать все точки в прямоугольник $x \times y$. Давайте поймем, где должен находиться его правый верхний угол. Очевидно, его x-координата должна быть хотя бы X (иначе будет невозможно поместить точку с первой координатой X в прямоугольник, т.к. при выполнении действия над ней ее первая координата будет только увеличиваться).Аналогично, его y-координата должна быть хотя бы Y. Тогда для начала положим координаты правого верхнего угла как (X, Y) — то есть минимально возможные.

Как же научиться минимизировать количество действий? На самом деле, для этого нужно выбрать минимально возможные координаты x и y для правого верхнего угла (то есть, чтобы их нельзя было уменьшить).

Если все точки уже находятся в прямоугольнике или могут быть туда доставлены, то это и есть оптимальное расположение. Если же это не так, то какие точки могут являться проблемой (то есть не могут попасть в текущий прямоугольник)? Если такие есть, то ими являются точки с минимальной или максимальной диагональю. Как понять, что точка с максимальной диагональю не попадает в прямоугольник? Нужно посмотреть, что ее номер диагонали больше, чем максимальный номер диагонали рассматриваемого прямоугольника: $D_{max} > (Y - X) + x$. Если это так, то давайте подвинем прямоугольник на $\Delta Y = D_{max} - (Y - X) - x$ вверх ($Y := Y + \Delta Y$). Понятно, что это необходимое условие для того, чтобы точка с максимальной диагональю попала в прямоугольник.

Аналогичное рассуждение для точки с минимальной диагональю: если $D_{min} < (Y - X) - y$, то подвинем прямоугольник на $\Delta X = D_{min} - (Y - X) + y$ вправо ($X := X + \Delta X$). Также понятно, что это необходимое условие, то есть без него бы мы не смогли получить какой-либо ответ.

Заметим, что из верхних двух действий мы выполним не более одного, то есть не может быть одновременно:

$$D_{max} > (Y - X) + x \tag{1}$$
$$D_{min} < (Y - X) - y \tag{2}$$

Так как в таком случае при домножении (2) на -1 и сложении двух неравенств получится

$$D_{max} - D_{min} > (Y - X) + x - (Y - X) + y = x + y$$

$$D_{max} - D_{min} > x + y$$

А такой случай мы отсекли в самом начале.

Значит, в самом начале оба условия выполняться не могли. Также несложно показать, что после увеличения X первое условие не могло начать выполняться ($D_{max} > (Y - X) + x \geqslant (Y - (X + \Delta X)) + x$). Аналогично первое условие не могло выполниться после увеличения Y. Значит, мы сделали необходимое изменение координаты. Очевидно, что оно достаточное: понятно, что если номер диагонали вершины попадает в нужный прямоугольник, то сама вершина может в него попасть: в противном случае она находится выше/правее его — что невозможно, ведь изначально мы выбирали X и Y как максимальные координаты.

Таким образом, мы нашли координаты правого верхнего угла прямоугольника. Осталось посчитать количество действий: понятно, что для конкретной точки (x_i, y_i) минимальное количество действий, чтобы попасть в прямоугольник, это $\max(0, X - x[i] - x, Y - Y[i] - y)$. Осталось посчитать сумму этой величины по всем точкам.

Получаем решение задачи за $\mathcal{O}(n)$ на тестовый случай.

Задача E. По классике

Автор задачи: Федор Ушаков, разработчик: Михаил Первеев

Для начала построим массив $q_1, q_2, \ldots, q_n$, где q_i — позиция элемента i в итоговом массиве. Существует множество способов построить данный массив.

Например, можно воспользоваться деревом отрезков. Будем поддерживать дерево отрезков над массивом t длины n, i-й элемент которого будет хранить 0, если позиция i итогового массива еще не занята, и 1 в противном случае. Далее мы будем обрабатывать запросы добавления с конца. Изначально присвоим 0 всем элементам массива t.

Пусть мы хотим обработать добавление числа i. Для того, чтобы понять его позицию в итоговом массиве, необходимо найти p_i-й ноль в массиве t. Позиция этого нуля и будет позицией числа i в итоговом массиве. После этого необходимо заменить 0 на 1, а также присвоить в q_i позицию найденного нуля. Поиск k-го нуля и изменение элемента массива можно выполнять при помощи дерева отрезков за $\mathcal{O}(\log n)$.

В случае использования языка C++, можно воспользоваться структурой данных `__gnu_pbds::tree` (эта структура данных также известна как `ordered_set`) вместо дерева отрезков.

Теперь, когда мы построили массив q, можно приступить к решению задачи. Рассмотрим некоторый момент времени, когда в массив были добавлены числа $1, 2, \ldots, i$. Рассмотрим некоторую последовательность чисел $x_1, x_2, \ldots, x_k$, такую что $1 \leqslant x_1 < x_2 < \ldots < x_k \leqslant i$. Данная последовательность будет возрастающей подпоследовательностью массива тогда и только тогда, когда ее элементы будут присутствовать в массиве именно в таком порядке. Иными словами, должно выполняться неравенство $q_{x_1} < q_{x_2} < \ldots < q_{x_k}$. Таким образом длина НВП массива после добавления первых i чисел равна длине НВП массива $q_1, q_2, \ldots, q_i$. Теперь задача свелась к тому, чтобы найти НВП на каждом префиксе построенного массива q.

Для того, чтобы сделать это, воспользуемся классическим решением задачи нахождения НВП за $\mathcal{O}(n \log n)$. Будем добавлять элементы по одному слева-направо, поддерживая массив $dp[i]$, где $dp[i]$ — минимальное значение элемента, на который может оканчиваться возрастающая подпоследовательность длины i. Также будем поддерживать текущий ответ — максимальную длину возрастающей подпоследовательности, которую мы можем получить.

Пусть мы рассмотрели элементы $q_1, q_2, \ldots, q_{i-1}$, поддерживая массив dp, и текущая длина НВП равна k. Добавим в рассмотрение элемент q_i. Для этого при помощи двоичного поиска найдем минимальное число j, такое что $dp[j] > q_i$. После этого в случае, если $dp[j-1] < q_i$, присвоим $dp[j]$ значение q_i. Наконец, если значение $dp[j]$ было обновлено, присвоим переменной k значение $\max(k, j)$.

Таким образом, обе части решения работают за $\mathcal{O}(n \log n)$.

Задача F. Обмен и удаление

Автор и разработчик задачи: Валерий Родионов

На каждом шаге последний элемент прыгает влево и затирает какой-то из предыдущих элементов. Заметим, что элементы $1, 2, \ldots, n-k$ никогда не прыгают, поэтому их относительный порядок сохраняется. Посмотрим, какие из удалятся после k шагов. Это будут элементы $n-k, n-k-1, \ldots, n-k-l+1$ (то есть какой-то суффикс), так как в противном случае последовательность не будет возрастающей. Эти элементы будут затерты некоторыми l элементами от $n-k+1$ до n. Зафиксируем множество из l элементов от $n-k+1$ до n, которые останутся после k шагов. Назовем их хорошими, а остальные $k-l$ назовем плохими. Зафиксируем для каждого хорошего элемента его финальную позицию (то есть какой из элементов $n-k, n-k-1, \ldots, n-k-l+1$) он затрет. Плохие элементы неразличимы, поэтому можно считать, что они не прыгают, а просто выбирают один из плохих элементов слева или финальную позицию какого-то хорошего элемента для прыжка, но сами остаются на месте.

Всего есть $k-l$ плохих элементов. Пусть i-й ($1 \leqslant i \leqslant k-l$) плохой элемент равен b_i. Выберем для каждого плохого элемента значение p_i, которые является либо позицией от $n-k+1$ до b_i, либо некоторым хорошим элементом, большим b_i. Построим биекцию между последовательностями $p_1, p_2, \ldots, p_{k-l}$ и последовательностями удалений, ставящими хорошие элементы на установленные финальные позиции, следующим образом:

Пусть есть последовательность $p_1, p_2, \ldots, p_{k-l}$. Сделаем k шагов. На каждом шаге будем делать следующее. Пусть последний элемент является хорошим. Тогда, если он выбран как p_i для некоторого плохого элемента, то он перепригивает на позицию самого правого из таких плохих элементов. Иначе он перепрыгивает на свою финальную позицию. Пусть последний элемент является плохим. Тогда он выбрал в качестве p_i некоторую позицию левее себя (так как в противном случае он бы уже затерся каким-то хорошим элементом). Если на позиции p_i сейчас стоит хороший элемент, то перепрыгнем в финальную позицию этого хорошего элемента, иначе просто перепрыгнем в позицию p_i.

Легко видеть, что таким образом можно получить любую последовательность удалений, которая ставит все хорошие элементы на свои финальные позиции.

Количество способов выбрать p_i равно $(k-l+i)$ (эквивалентно выбору любого хорошего элемента или плохого элемента слева), поэтому общее число способов равно $k \cdot (k-1) \cdot \ldots \cdot (k-l+1) = k!/(k-l)!$.

Также нужно домножить на $l!$ (выбор финальных позиций хороших элементов) и $\binom{k}{l}$ (выбор хороших). Получаем, что ответ равен сумме по l от 0 до $min(k, n-k)$ значений $k!/(k-l)! \cdot l! \cdot \binom{k}{l}$).

Задача G. Трасса M-11

Авторы задачи: Александра Олемская, Александр Понкратов, разработчик: Александр Понкратов

Рассмторим все точки типа 1 и запомним сколько есть точек типа 0 есть левее её, пусть их cnt_i. Далее для каждой точки типа 1 найдем самую удаленную точку от текущей такую, что расстояние между ними не превосходило k, пусть её индекс j тогда все точки типа 1 от j до $i-1$ потенциально образуют удобную тройку с точкой i, а именно $cnt_i - cnt_p$ ($j \leqslant p < i$) троек. Тогда к ответу надо прибавить $\sum_{p=j}^{i-1} cnt_i - cntp = (i-j) * cnt_i - \sum_{p=j}^{i-1} cnt_p$. Искать самую удаленную точку можно бинарным поиском или двумя указателями, а считать вклад в ответ для текущей рассматриваемой точки с помощью префиксных сумм. В зависимости от реализации полуачаем решение за $\mathcal{O}(n)$ или $\mathcal{O}(n \log n)$.

Задача H. Роботы-исследователи

Автор и разработчик задачи: Александр Бабин

Так как роботы знают свое положение относительно друг друга, то они могут посетить все поля, которые находятся между ними. После того, как они сделают это, будет посещен какой-то отрезок $[\min\{x, y\}, \max\{x, y\}]$ полей между ними. Далее этот отрезок может расширяться на одно поле по левой и правой границе.

Назовем строку *и бордером* строки s, если u одновременно является и суффиксом, и префиксом строки s. Утверждается, что если какой-то эксперимент закончился тем, что роботы посетили отрезок полей $[l, r]$, то строка $s[l \dots r]$ является бордером строки s.

При этом, если в какой-то момент роботы посетили отрезок полей $[l, r]$ такой, что $s[l \dots r]$ не является бордером, то $s[l \dots r]$ не является либо суффиксом, либо префиксом строки, таким образом роботы могут гарантировать расширение этого отрезка либо в левую, либо в правую сторону.

Поэтому можно переформулировать задачу следующим образом: для запроса (x, y) требуется найти отрезок минимальной длины $[l, r]$ такой, что $x, y \in [l, r]$, а так же $s[l \dots r]$ является бордером строки s.

Для того, чтобы сделать это за эффективное время отдельно найдем левую и правую границу. Опишем, как можно найти правую границу (левая ищется тем же самым способом, только предварительно надо развернуть строку s).

Вычислим массив $\pi_1, \dots, \pi_n$ — *префикс-функцию* строки s, то есть такой массив, что $\pi_1 = 0$ и π_k равняется длине наибольшего нетривиального бордера строки $s[1 \dots k]$. Тогда понятно, что $n, \pi[n], \pi[\pi[n]], \dots$ — это длины всех бордеров строки s. Далее за линейное время с помощью метода динамического программирования легко вычислить массив $q_1, \dots, q_n$, где q_k — наименьшее значение такое, что $s[q_k \dots k]$ — бордер строки s.

Рассмотрим запрос (x, y), для него требуется найти наименьшее $k \geqslant \max\{x, y\}$ такое, что $x, y \in [q_k, k]$, это и будет бордер с наименьшей правой границей, который содержит пару позиций x, y. Найти такое k для каждого запроса можно, например, в оффлайне, если отсортировать все запросы по возрастанию $\min\{x, y\}$. После этого множество S, состоящее из чисел k, таких что $q_k \leqslant \min\{x, y\}$ будет только расширяться, а это значит, что его можно поддерживать в `std::set` за суммарное время $O(n \log n)$. Далее, для того, чтобы ответить на запрос, достаточно воспользоваться методом `.lower_bound`. В итоге мы получаем алгоритм, который работает за время $O(n \log n)$.

Ремарка. Может показаться неправильным то, что мы ищем отдельно левую и правую границу в каждом из запросов. Небольшое математическое рассуждение ниже должно убедить читателя в обратном.

Лемма. Пусть $[l, r]$ и $[a, b]$ такие отрезки, что $s[l \dots r]$ и $s[a \dots b]$ — бордеры строки s, тогда $J = [l, r] \cap [a, b]$ также является бордером строки s. Это так, потому что $s[J]$ является суффиксом либо строки $s[l \dots r]$, либо $s[a \dots b]$, а также является префиксом одной из этих двух строк. То есть $s[J]$ является суффиксом какого-то бордера строки s, а значит является суффиксом и для строки s, аналогично $s[J]$ — это префикс строки s. Таким образом $s[J]$ является и префиксом, и суффиксом s, значит $s[J]$ — это бордер строки s.

Пусть $s[l \dots r]$ — минимальный бордер строки s, содержащий пару позиций (x, y), а $s[p \dots r']$ и $s[l' \dots q]$ — бордеры s, соответственно, с минимальной правой границей и максимальной левой границей, которые содержат пару позиций (x, y). Тогда $l \leqslant l' \leqslant r' \leqslant r$ ввиду определения $s[p \dots r']$ и $s[l' \dots q]$. Но также $s[l' \dots r']$ — бордер s ввиду леммы. Но в виду минимальности длины $[l, r]$ и вложенности $[l', r'] \subseteq [l, r]$ имеем $[l', r'] = [l, r]$, что и требовалось доказать.

Задача I. Шалость

Автор и разработчик задачи: Николай Ведерников

Заведём стек и два указателя по одному в каждой строке. Если символы в строках по указателям совпадают и на стеке нет букв, то мы считаем, что эта буква сохранилась из начальной строки (и сдвигаем оба указателя вперёд). Иначе возможны две ситуации:

- в вершине стека лежит точно такая же буква. Значит, можно считать, что эти две одинаковые буквы поставили вместе, так остальные парные буквы, если они были, мы удалили до этого. Удалим букву со стека и подвинем указатель в строке s_2.
- стек пустой или в вершине лежит другая буква. Значит, положим эту букву в стек и передвинем указатель для s_2.

Если мы рассмотрели все символы строки s_1 (можно проверить, что указатель достиг конца строки) и стек в итоге остался пустым, то считаем, что из строки s_1 можно получить строку s_2 путём применения `шалостей`, иначе нет.

Задача J. Кошмарная сумма

Автор и разработчик задачи: Евгений Пахомов

Пусть $C = 300\,000$. Так как все элементы массива различны, суммарное число пар $(\min, \texttt{частное})$ составляет $O(C \log C)$ (сумма гармонического ряда).

Зафиксируем i — позицию минимума. Заметим, что частное $\left\lfloor \frac{\max}{\min} \right\rfloor$ может принимать только значения $d = 1, 2, \ldots, \left\lfloor \frac{C}{a_i} \right\rfloor$. Как уже было отмечено, все эти значения можно честно перебрать. Пусть мы знаем i и d. Какие отрезки нам подойдут?

Минимумом на отрезке должно быть a_i, поэтому $l \leqslant i \leqslant r$ и на отрезке не должно быть чисел, меньших a_i. Чтобы этого добиться, можно пройтись слева направо и справа налево по массиву, поддерживая стек минимумов. Тогда для всех i мы сможем насчитать ближайшее число в массиве справа/слева от i, меньшее a_i.

Осталось понять, каким должен быть максимум на отрезке. Так как мы зафиксировали d, максимум должен по значению лежать в отрезке $[a_i \cdot d; a_i \cdot d + a_i - 1]$. Какие понять, какие ограничение это условие накладывает на l и r? На отрезке не должно быть чисел, больших или равных $a_i \cdot (d+1)$, поэтому l строго больше индекса ближайшего слева к i числа, большего или равного $a_i \cdot (d+1)$. Аналогично r должно быть строго меньше индекса ближайшего справа к i числа, большего или равного $a_i \cdot (d+1)$. Кроме этого, накладываются ограничения на l и r с другой стороны: максимум должен быть не менее $a_i \cdot d$, то есть l должно быть не больше, чем индекс ближайшего слева к i числа, большего или равного $a_i \cdot d$. Аналогично r должно быть не меньше, чем индекс ближайшего справа к i числа, большего или равного $a_i \cdot d$.

Искать ближайшее слева/справа число, большее или равное x можно при помощи спуска по дереву отрезков или бинпоиска на стеке максимумов. Значит, подсчет указанных ограничений на l и r затратит $O(n \log n \log C)$ времени.

Значит, имеем для конкретной пары (i, d) ограничения вида: $x < l \leqslant q$ и $p \leqslant r < y$. Как пересчитать ответ?

Пусть $dl = q - x$, $dr = y - p$. Мы хотим прибавить к ответу $d \cdot \texttt{кол-во хороших отрезков}$. Это значение несложно вычислить при помощи формулы включений-исключений:

$$\texttt{кол-во хороших отрезков} = dl \cdot (y - i) + dr \cdot (i - x) - dl \cdot dr$$

Таким образом, асимптотика итогового решения составит $O(n \log n \log C)$.

Задача K. Криптография Пети

Автор задачи: Демид Кучеренко, разработчик: Владимир Рябчун

Любое дерево с $n \geqslant 2$ вершинами можно построить следующим образом. Начнём с одного ребра. На каждом шаге будем выбирать какой-то лист v и число $i \geqslant 1$, после чего подвешивать i вершин к этому листу. При этом количество путей длины 2 в дереве увеличится на $\frac{i(i+1)}{2}$.

Для решения данной задачи посчитаем величину $dp[n][s]$ — можно ли построить дерево из n вершин с s путями длины 2.

$$dp[n][s] = \bigvee_{1 \leqslant i \leqslant n} dp[n-i][s - \frac{i(i+1)}{2}]$$

Для решения необходимо вычислять динамику лениво, при данных ограничениях для любого n будет посещено порядка 2000 значений s.

Задача L. Два самоката

Автор задачи: Николай Ведерников, разработчик: Екатерина Ведерникова

Для решения задачи нужно посчитать два значения и сравнить их. Для фирмы `W` нужно t нацело поделить на 60, затем полученное целое число умножить на 60. Полученное количество секунд нужно умножить на цену c_1, чтобы получить стоимость поездки на самокате фирмы `W` в копейках.

Для фирмы `Y` t нужно умножить на цену c_2. После этого нужно полученное число поделить на 100, если при делении возникает остаток, к частному нужно прибавить 1. Полученное целое количество рублей нужно умножить на 100, чтобы также получить стоимость в копейках.