Задача A. Лягушки

Пусть $a < b < c$. Тогда, если $a + 1 = b$ и $b + 1 = c$, то минимальное и максимальное число прыжков — ноль, так как из данного положения лягушки прыгать не могут.

Если $a + 2 = b$ или $b + 2 = c$, то третья лягушка может прыгнуть в позицию между ними и образовать стабильное положение, то есть минимальное число прыжков — 1.

Иначе, лягушки всегда могут закончить прыгать за 2 прыжка. Пусть в начале лягушка из позиции a прыгнет в позицию $b + 2$, а затем лягушка из позиции c прыгнет в позицию $b + 1$. Если $b + 1 = c$, то лягушек можно развернуть и совершить аналогичные прыжки.

Стратегия для максимального числа прыжков выглядит следующим образом: пусть изначально лягушка с позиции a прыгнет на позицию $b + 1$ или же лягушка с позиции c прыгнет на позицию $b - 1$. Затем, две лягушки будут находиться подряд с одного краю, а другая — на противоположной стороне. Можно сделать так, чтобы крайняя лягушка прыгала через соседнюю, таким образом ответ будет $max(b - a - 1, c - b - 1)$. Лучше ответ получить нельзя, так как изначально мы должны были совершить какой-то ход, и любой ход уменьшает диаметр множества хотя бы на 1.

Задача B. Автомобиль «Пантера»

Переберем исходное время t. Можно заметить, что при любом s оно будет порядка $O(d^{0.5})$.

Оптимальной стратегией движения автомобиля будет следующая: в начале мы ускоряемся, затем не меняем скорость, а в конце замедляемся. Найдём максимальное расстояние, которое можно проехать за время t. Заметим, что если мы смогли проехать расстояние $d_1 > d$, то d мы тоже сможем проехать(достаточно уменьшать скорость на 1 в каких-то точках, единственное ограничение — модуль разности скоростей в соседние моменты времени не должен быть больше 1). В тупом решении можно просто симулировать за $O(t)$, но можно и посчитать расстояние за $O(1)$: определим максимальную скорость x, до которой можно ускориться, чтобы потом успеть замедлиться до s; затем достаточно посчитать сумму от 1 до x, умножить x на время, которое мы будем двигаться с этой скоростью, и посчитать сумму от x до s, если $x > st$. Таким образом, получаем решение за $O(d^{0.5})$.

Для полного решения достаточно заметить, что t можно перебирать с помощью бинарного поиска по ответу, так как значения функции «максимальное расстояние, которое можно проехать за время x» возрастают, и получить решение за $O(log_2(d))$.

Также существует решение за $O(1)$: можно расписать формулу максимального пройденного расстояния за время t, и найти оптимальное t, решив квадратное уравнение.

Задача BZ. Автомобиль «Пантера»

Переберем исходное время t. Можно заметить, что при любом $SpeedLimit$ оно будет порядка $O(d^{0.5})$.

Оптимальной стратегией движения автомобиля будет следующая: в начале мы ускоряемся, затем не меняем скорость, а в конце замедляемся. Найдём максимальное расстояние, которое можно проехать за время t. Заметим, что если мы смогли проехать расстояние $d_1 > d$, то d мы тоже сможем проехать(достаточно уменьшать скорость на 1 в каких-то точках, единственное ограничение — модуль разности скоростей в соседние моменты времени не должен быть больше 1). В тупом решении можно просто симулировать за $O(t)$, но можно и посчитать расстояние за $O(1)$: определим максимальную скорость x, до которой можно ускориться, чтобы потом успеть замедлиться до 0; затем достаточно посчитать сумму от 1 до x, умножить x на время, которое мы будем двигаться с этой скоростью, и посчитать сумму от x до 1. Таким образом, получаем решение за $O(d^{0.5})$.

Для полного решения достаточно заметить, что t можно перебирать с помощью бинарного поиска по ответу, так как значения функции «максимальное расстояние, которое можно проехать за время x» возрастают, и получить решение за $O(logd)$.

Также существует решение за $O(1)$: можно расписать формулу максимального пройденного расстояния за время t, и найти оптимальное t, решив квадратное уравнение.

Задача C. 2022

Для решения за $O(n^4)$ можно просто перебрать все четверки $1 \leqslant i, j, k, h \leqslant n$ и проверить для них условие.

Чтобы решить задачу за $O(n^3)$, предпосчитаем cnt_x – количество элементов, которые равны x. Теперь можно перебирать только тройки $(i,\ j,\ k)$. При фиксированных $(i,\ j,\ k)$ четвертое число определяется однозначно, поэтому к ответу нужно добавить $cnt_{2022-a_i-a_j-a_k}$.

Для решения за $O(n^2)$ предпосчитаем $cnt2_x$ – количество пар $1 \leqslant i, j \leqslant n$ таких, что $a_i + a_j = x$, перебрав все пары. В одну четверку входят две пары. Если зафиксировать сумму чисел в первой паре, то сумма во второй определяется однозначно, поэтому можно перебрать сумму s в первой от 0 до 2022 и прибавить к ответу $cnt2_s \cdot cnt2_{2022-s}$.

Чтобы сделать предпосчет быстрее $O(n^2)$ и получить полное решение, заметим, что $cnt2$ можно получить из cnt, перебрав значение первого числа в паре, так как $cnt2_x = cnt_0 \cdot cnt_x + cnt_1 \cdot cnt_{x-1} + \ldots + cnt_x \cdot cnt_0$.

Задача CZ. 2022

Для решения за $O(n^2)$ можно просто перебрать все пары $1 \leqslant i < j \leqslant n$ и проверить для них условие на сумму.

Чтобы решить задачу за $O(n)$, будем перебирать индексы в порядке возрастания и поддерживать массив cnt_x — количество рассмотренных чисел со значением x. При рассмотрении индекса i добавим к ответу все пары, в которых i справа. У левых чисел в таких парах значение определяется однозначно и равно $2022 - a_i$, значит к ответу нужно прибавить cnt_{2022-a_i}. Еще нужно увеличить cnt_{a_i} на 1.

Задача D. Максимальный XOR

С помощью сортировки пузырьком можно получить любую перестановку цифр в обоих числах, меняя соседние. Это же верно и для итогового XOR, потому что если поменять соседние цифры на одинаковых позициях сразу в обоих числах, то они поменяются и в их XOR. Таким образом, получив какой-то XOR, мы всегда можем отсортировать в нём цифры по невозрастанию. Поэтому нам нужно получить как можно больше цифр F, дальше как можно больше E и так далее.

Чтобы решить задачу для $n \leqslant 8$, можно просто перебрать все перестановки цифр в первом числе, посчитать XOR со вторым, отсортировать в нём цифры невозрастанию и выбрать из таких XOR-ов наибольший.

Если a состоит только из нулей, то ответом будут цифры b, расположенные по невозрастанию.

Можно заметить, что в тестах, где a состоит только из цифр $0, 1, 2, 3$ и b только из $0, 4, 8, C$, записи a и b никогда не пересекаются по битам, поэтому оптимально отсортировать цифры a и b по невозрастанию.

Для следующих решений предпосчитаем da_i и db_i – количество цифр со значением i в a и в b соответственно.

Когда оба числа состоят только из цифр 0 и 1, нужно получить как можно больше единиц в итоговом XOR. Тогда в ответе будет префикс из $min(da_0, db_1) + min(da_1, db_0)$ единиц и суффикс из нулей.

Пусть в a только цифры со значениями w, v и в b только со значениями x, y. Тогда можно перебрать m – количество позиций, в которых стоят w и x. После чего через da_w, da_v, db_x, db_y и m однозначно вычисляется количество позиций с w и y, v и x, v и y. Тогда мы знаем, сколько цифр с каким значением в XOR сейчас, и можем выбрать лучший из результатов для разных m, сравнивая количества цифр $F, E, \ldots 0$ в получающихся XOR-х.

Чтобы построить общее решение, будем перебирать цифры в результате от F до 0. Пусть мы уже рассмотрели все цифры больше d. Теперь нам нужно получить как можно больше цифр d, чтобы максимизировать ответ. Для этого переберем цифру x в первом числе. Тогда $x\ \ XOR\ \ d$ — цифра из второго числа, дающая нужный XOR в паре с x. Будем приписывать к ответу d, пока $da_x > 0$ и $db_{x\ \ XOR\ \ d} > 0$ и уменьшать da_x и $db_{x\ \ XOR\ \ d} > 0$ на 1.

Задача DZ. Максимальный XOR

С помощью сортировки пузырьком можно получить любую перестановку блоков в обоих числах, меняя соседние. Это же верно и для итогового XOR. Потому что если мы смогли получить какой-то XOR, то можно менять в нем соседние блоки местами, просто меняя соответсвующие соседние

блоки в числах a и b на тех же позициях. В частности, получив какой-то XOR, мы всегда можем отсортировать в нём блоки по невозрастанию.

Чтобы решить задачу для $n \leqslant 8$, можно просто перебрать все перестановки блоков в первом числе, посчитать XOR со вторым, отсортировать в нем блоки по невозрастанию и выбрать из таких XOR-ов наибольший.

Если a состоит только из нулей, то ответом будут просто блоки b, расположенные от больших к меньшим.

Можно заметить, что в тестах, где a состоит только из блоков 00 и 01 и b только из блоков 00, 10, записи a и b никогда не пересекаются по битам, поэтому оптимально отсортировать блоки a и b по невозрастанию.

Для следующих решений предпосчитаем da_{ij} и db_{ij} – количество блоков ij, $0 \leqslant i, j \leqslant 1$ в a и в b соответсвенно.

Когда оба числа состоят только из блоков 00 и 01, нужно получить как можно больше блоков 01 в XOR. Тогда в ответе будет префикс из $min(da_{00}, db_{01}) + min(da_{01}, db_{10})$ блоков 01 и суффикс из нулей.

Мы знаем, что блоки в полученном XOR можно отсортировать по невозрастанию. Значит для оптимального решения нам нужно получить как можно больше блоков 11 и поставить их в начало XOR. Среди решений с максимальным количеством 11 нужно выбрать то, в котором больше всего блоков 10 и расставить их после блоков 11. Аналогично среди всех таких решений нужно выбрать то, в котором больше всего блоков 01 и поставить их после блоков 10.

Чтобы построить общее решение, будем перебирать значение блоков в порядке 11, 10, 01, 00. Пусть мы уже рассмотрели все блоки до ij. Тогда сейчас нам нужно получить как можно больше блоков ij, чтобы максимизировать ответ. Для этого переберем значение блока xy в первом числе. Тогда $xy \ \ XOR \ \ ij$ — значение блока из второго числа, дающего нужный XOR в паре с xy. Будем приписывать к ответу ij, пока $da_{xy} > 0$ и $db_{xy \ \ XOR \ \ ij} > 0$ и уменьшать da_{xy} и $db_{xy \ \ XOR \ \ ij} > 0$ на 1.

Задача E. Массив сумм

Заметим, что если число a не заканчивается на 9, то сумма цифр в a на 1 меньше суммы цифр в $a + 1$. В ином случае суффикс из цифр 9 превращается в нули, и сумма цифр уменьшается на количество девяток на суффиксе, умноженное на 9, и увеличивается на 1. Так мы узнаем, что у числа $a + i$ на суффиксе k девяток. Это дает нам информацию о первых k цифрах числа a, поэтому нужно посмотреть на максимальную разницу $sum_i - sum_{i+1}$ по всем i.

Остается лишь подобрать число a подходящее всем критериям и добавить в его начало цифр, чтобы сумма a совпадала с sum_1.

Задача F. Возрастающие пути

Вначале заметим, что нам достаточно рассматривать только простые x. Из этого следует, что длина любого возрастающего пути не более 23, так как максимальный вес ребра 10^7.

Для решения подзадачи 1 нужно перебрать все пути и найти найти на них ответ. Будем вначале перебирать первую вершину пути, переберём первое ребро и один из простых делителей веса этого ребра, запустим dfs от второго конца ребра, переходя из вершин в следующие так, чтобы путь оставался простым и вес следующего ребра делился на нужную степень зафиксированного простого числа. Решение этой подзадачи работает за $O(n^2 * logMAXW)$ или $O(n^2)$ в зависимости от реализации.

В подзадаче 2 граф — цепочка из n вершин. В этой подзадаче достаточно также перебрать первую вершину, ребро выходящее из неё и простой делитель веса этого ребра. А далее запустить dfs от второго конца ребра. Заметим, что всего для одной вершины, выбранного ребра и простого будет проверено не более 24 вершин, так как длина возрастающего пути всегда не более 23. Решение этой подзадачи работает за $O(n * log^2 MAXW)$.

Для полного решения давайте разложим вес каждого ребра на простые при помощи линейного решета Эратосфена для нахождения минимального простого делителя для каждого числа от 1 до 10^7. Подвесим граф за вершину 1. После переберём все простые числа, каждое из

которых является простым делителем хотя бы для веса одного из рёбер. Зафиксируем тогда какое-то простое число p и оставим только те рёбра, вес которых делится на это простое число p. Тогда у нас останутся вершины, которые соединены хотя бы с одним из оставшихся рёбер. Посчитаем для каждой такой вершины две динамики: $up[v]$ — максимальная длина возрастающего пути, приходящего из какого-то ребёнка вершины v, $down[v][k]$ — максимальная длина пути, идущего в одного из детей вершины v, такого, что вес первого ребра делится на p^k, вес второго ребра делится на p^{k+1}, ... , вес m-го ребра делится на p^{k+m-1}. Тогда научимся пересчитывать динамики для вершины v через детей: вершина u_1 и вес ребра (v, u_1) делится на p^{c_1}, вершина u_2 и вес ребра (v, u_2) делится на p^{c_2}, ... ,вершина u_d и вес ребра (v, u_d) делится на p^{c_d}. Поэтому $up[v] = max(min(c_1, up[u_1] + 1), min(c_2, up[u_2] + 1), \ldots, min(c_d, up[u_d] + 1))$, а $down[v][k] = max(down[u_i][k + 1] + 1)$ для всех i таких, что $c_i \geqslant k$. Тогда ответ на задачу нужно обновить с $up[v], down[v][1]$ и перебрать все такие i и k, что $c_i >= k$, и существует j, что $u_i \neq u_j$ и $up[u_j] >= k - 1$. Тогда ответ нужно обновить с $down[u_i][k + 1] + k$. Для одного простого числа мы можем всё посчитать за $O(E(p) * logMAXW)$, где $E(p)$ — количество рёбер, вес которых делится на p и $MAXW$ — максимальный вес ребра. Суммарно для всех простых мы обработаем за $O(n * MAXD * logMAXW)$, где $MAXD$ - максимального количество простых делителей у веса ребра. Заметим, что $MAXD$ не более $logMAXW$, тогда всё решение работает за $O(n * log^2 MAXW + MAXW)$.

Для решения подзадачи 3 достаточно посчитать динамики только для простого числа 2. Решение этой подзадачи работает за $O(n * logMAXW)$.

Для решения подзадачи 4 вместо линейного решета Эратосфена достаточно для каждого ребра отдельно разложить вес на простые числа. Решение этой подзадачи работает за $O(n * \sqrt{MAXW} + n * log^2 MAXW)$.

Вместо двух динамик для каждого простого числа p можно действовать итеративно. Оставим также в графе только рёбра, вес которых кратен p. На итерации i будем хранить рёбра, на которые могут заканчиваться возрастающие пути длины i. Тогда, чтобы перейти с итерации i на $i + 1$, достаточно перебрать рёбра из вершин, которые являются концами одного из рёбер итерации i, и проверить, что вес ребра кратен p^{i+1}, а также, что это ребро не ломает простоту для какого-то пути. Заметим, что i-я итерация работает за количество рёбер, вес которых делится на p^i. Тогда суммарно решение работает за сумму степеней простых чисел в разложении всех весов рёбер. Для одного ребра сумма степеней $O(logMAXW)$. Значит для всех рёбер суммарно работает за $O(n * logMAXW + MAXW)$.

Задача G. Башни 2.0

В первой подзадаче можно было явно построить граф, где ребро (u, v) проведено если из u-й башни можно прыгнуть на v-ю. После этого достаточно из каждой вершины запустить DFS и посчитать количество посещённых вершин. Пусть m – количество рёбер в графе, тогда n DFS-ов работют за $O(nm)$. Поскольку $m = \Theta(n^2)$ (нижняя оценка когда высоты башен возрастают), алгоритм работает за $O(n^3)$ и проходит при ограничениях $n \leqslant 100$.

Для решения второй подзадачи мы можем для каждой вершины хранить битсет достижимых вершин. Более формально, битсет вершины v имеет размер n, и i-й бит равен 1 если вершина i достижима из вершины v, 0 иначе. Сначала научимся эффективно строить граф. Заметим, что ребро (u, v) существует тогда и только тогда, когда v-я башня является одним из максимумов на отрезке, в котором u — середина, а v является одним из концов. Мы можем предпосчитать максимумы на каждом отрезке за $O(n^2)$, после этого можно проверить существование ребра за $O(1)$. Поскольку все башни имеют разную высоту, если мы будем рассматривать их в порядке убывания высоты, все рёбра из очередной башни будут вести в предыдущие. Это значит, мы можем поддерживать инвариант что множество достижимых из всех рассмотренных вершин посчитано корректно, после чего для вычисления множества достижимых из очередной вершины v достаточно взять OR битсетов вершин, куда ведут рёбра из v. Асимптотика такого решения — $O\left(\frac{n^3}{w}\right)$, где w — длина машинного слова.

В третьей подзадаче все высоты равны либо 1, либо 2. Из любой вершины существует рёбра во все вершины со значением 2, поэтому их колчиество можно добавить к ответу для всех вершин. Кроме этого, нам нужно посчитать для каждой вершины v со значением 1 количество других вершин

со значением 1, которые из неё достижимы. Можно заметить, что если рядом с v в массиве есть значение 2, мы не сможем попасть из v в другую вершину со значением 1. В противном случае, мы можем, используя рёбра длины 1, добраться до всех вершин в максимальном по включению отрезке из 1, содержащем v, и ни одна другая вершина со значением 1 не достижима. Используя эти наблюдения, можно получить линейное решение.

Четвёртая подзадача была идентична третьей, но у нас дополнительно могли быть башни высотой 3. Сначала заметим, что все такие башни достижимы из всех остальных. Для каждой башни высотой 2 предпосчитаем следующую башню высоты 2 слева и справа, а также ближайшую башню высоты 3. Пусть башни u и v — башни высоты 2, и существует ребро из u в v. Пусть $w_1, w_2, \ldots, w_k$ — башни высоты 2 на отрезке от u до v. Заметим, что вместо ребра от u до v, мы могли бы пойти по пути $u \to w_1 \to w_2 \to \ldots \to w_k \to v$. Это значит, что из всех рёбер между башнями высоты 2 можно оставить только те, которые ведут в ближайшую слева и справа башню, которые мы предпосчитали. Это позволяет легко посчитать для каждой вершины v высоты 2 значения l_v — количество достижимых слева, и r_v — справа, а также итоговый ответ. Теперь рассмотрим башни v высоты 1. Используя критерий из предыдущей подзадачи, мы можем понять какое множество башен высоты 1 достижимо — это может быть либо только исходная башня, либо максимальный по включению отрезок башен высоты 1, обозначим его за $[a, b]$. Нам осталось посчитать количество достижимых башен высоты 2. Заметим, что в любом пути из v первая башня высоты больше 1 должна быть либо $a-1$-й, либо $b+1$-й. Тогда достаточно добавить к ответу l_{a-1} если вершина $a-1$ достижма и r_{b+1} если вершина $b+1$ достижима. Решение работает за $O(n)$.

Сделаем наблюдение, полезное для полного решения. Мы хотим понять из каких вершин есть ребро в вершину v. Пусть gl_v — ближайший элемент в массиве слева от v, больший v, gr_v — аналогичный элемент справа. Если к некоторой вершине u есть более близкий чем v элемент, больший v-го, его можно выбрать из множества $\{gl_v, gr_v\}$. Это значит, что множество вершин, из которых проведено ребро в v — отрезок $\left[\left\lfloor \frac{gl_v+v}{2} \right\rfloor + 1, \left\lceil \frac{gr_v+v}{2} \right\rceil - 1\right]$.

Используя этот факт, можно получить решение пятой подзадачи. Пусть в вершину v ведут рёбра с отрезка $[c_v, d_v]$. Мы переберём v, посчитаем отрезок вершин, из которых достижима v, и добавим 1 к ответу на этом отрезке. Чтобы посчитать отрезок вершин, откуда достижима v, будем хранить текущий ответ (изначально это $[c_v, d_v]$), и отрезок рассмотренных вершин (изначально это $[v, v]$). После этого, пока отрезок рассмотренных вершин и ответ не совпадут, будем расширять отрезок рассмотренных вершин на 1 (слева или справа). Пусть u — вершина, которую мы только что добавили в отрезок рассмотренных, она должна принадлежать отрезку ответа, и мы можем объединить отрезок ответа с отрезком $[c_u, d_u]$. Когда отрезок ответа и рассмотренных вершин совпадут, не будет существовать вершин вне отрезка, из которой есть ребро в отрезок, следовательно, это все вершины, из которых достижима v. Решение работает за $O(n^2)$.

Для полного решения понадобится ещё несколько наблюдений. Заметим, что поскольку ребро (i, j) проведено тогда, когда j-й элемент является максимумом на отрезке, где i является серединой, а j – одним из концов, из наличия ребра (i, j) следует наличие рёбер (k, j) для $i < k < j$. Аналогичное утверждение можно сделать для ребра (i, j) когда $j < i$. Пусть в некотором пути есть два соседних ребра (u, v) и (v, w), такие, что $w < u < v$. Тогда, используя свойство выше, мы получаем, что есть ребро (u, w). Делая такие замены для произвольного пути, можно добиться того, чтобы все рёбра пути вели либо только налево, либо только направо. Посчитаем кол-во достижимых по рёбрам налево из каждой вершины, аналогично по рёбрам направо, и, сложив результаты, получим ответ. Пусть L_i – самая правая вершина левее i, в которую есть ребро из i. Докажем, что можно оставить только рёбра $i \to L_i$, и количество достижимых слева не изменится. Действительно, пусть в каком-то пути мы пошли по ребру $i \to j$ и $j < L_i$. Мы знаем что тогда должно быть ребро $L_i \to j$, и $i \to j$ можно заменить на $i \to L_i$, $L_i \to j$. Делая такие замены, мы можем удалить все «плохие» рёбра. Тогда ответ для вершины i — это ответ для L_i плюс 1. Это значит, что для решения задачи достаточно уметь эффективно находить L_i для каждого i.

Мы умеем за линейное время посчитать gl_i и gr_i для всех i, используя стандратный алгоритм со стеком. Поскольку входящие рёбра в каждую вершину ведут из отрезка, который мы можем вычислить, задача о подсчёте L_i сводится к следующей: даны n отрезков $[l_i, r_i]$. Для каждой позиции нужно вычислить минимальный индекс отрезка, который её покрывает. Существует множество

решений этой задачи. Мы можем сделать сканлайн, который поддерживает set отрезков, которые покрывают данную позицию, тогда для каждого отрезка нужно создать событие «добавить i» в позции l_i и «удалить i» в позиции $r_i + 1$. Такое решение работает $O(n \log n)$. Также можно решать задачу с помощью дерева отрезков (тоже за $O(n \log n)$) или с помощью системы непересекающихся множеств (за $O(n \cdot \alpha(n))$).

BONUS: попробуйте решить задачу «Башни 3.0» из 9-го класса. У вас есть q запросов (u, v, l, r), необходимо отвечать количество вершин на отрезке $[l, r]$, которые достижимы как из u, так и из v.

Задача GD. Башни 3.0

В первой подзадаче можно было явно построить граф, где ребро (u, v) проведено, если из u-й башни можно прыгнуть на v-ю. После этого достаточно из каждой вершины запустить DFS и посчитать множество посещённых вершин. Пусть m – количество рёбер в графе, тогда n DFS-ов работают за $O(nm)$. Чтобы ответить на запрос, достаточно перебрать индекс вершины на отрезке $[l, r]$ и проверить, сколько из них подходят под условие, используя предпосчитанную достижимость. Решение работает за $O(nm + qn)$, поскольку $m = \Theta(n^2)$ (нижняя оценка когда высоты башен возрастают), алгоритм работает за $O(n^3 + qn)$ и проходит при ограничениях $n \leqslant 100$, $q \leqslant 1000$.

Для решения второй подзадачи мы можем для каждой вершины v хранить битсет b_v достижимых вершин. Более формально, битсет вершины v имеет размер n, и i-й бит равен 1, если вершина i достижима из вершины v, 0 иначе. Сначала научимся эффективно строить граф. Заметим, что ребро (u, v) существует тогда и только тогда, когда v-я башня является одним из максимумов на отрезке, в котором u — середина, а v является одним из концов. Мы можем предпосчитать максимумы на каждом отрезке за $O(n^2)$, после этого можно проверить существование ребра за $O(1)$. Поскольку все башни имеют разную высоту, если мы будем рассматривать их в порядке убывания высоты, все рёбра из очередной башни будут вести в предыдущие. Это значит, что мы можем поддерживать инвариант что множество достижимых из всех рассмотренных вершин посчитано корректно, после чего для вычисления множества достижимых из очередной вершины v достаточно взять OR битсетов вершин, куда ведут рёбра из v. После того, как мы посчитали все битсеты, мы хотим эффективно отвечать на запросы. Для каждого l создадим битсет $filterL_l$, который содержит 1 в позициях, которые не меньше l. Аналогично создадим битсеты $filtetR$. Тогда ответ на запрос (u, v, l, r) — это количество единиц в AND b_u, b_v, $filterL_l$, $filterR_r$. Асимптотика такого решения — $O\left(\frac{n^3}{w}\right)$, где w — длина машинного слова.

В следующих четырёх подзадачах для того, чтобы отвечать на запросы, достаточно посчитать количество достижимых из каждой вершины.

В третьей подзадаче все высоты равны либо 1, либо 2. Из любой вершины существуют рёбра во все вершины со значением 2, поэтому их количество можно добавить к ответу для всех вершин. Кроме этого, нам нужно посчитать для каждой вершины v со значением 1 количество других вершин со значением 1, которые из неё достижимы. Можно заметить, что если рядом с v в массиве есть значение 2, мы не сможем попасть из v в другую вершину со значением 1. В противном случае, мы можем, используя рёбра длины 1, добраться до всех вершин в максимальном по включению отрезке из 1, содержащем v, и ни одна другая вершина со значением 1 не достижима. Используя эти наблюдения, можно получить линейное решение.

Четвёртая подзадача была идентична третьей, но у нас дополнительно могли быть башни высотой 3. Сначала заметим, что все такие башни достижимы из всех остальных. Для каждой башни высотой 2 предпосчитаем следующую башню высоты 2 слева и справа, а также ближайшую башню высоты 3. Пусть башни u и v — башни высоты 2, и существует ребро из u в v. Пусть $w_1, w_2, \ldots, w_k$ — башни высоты 2 на отрезке от u до v. Заметим, что вместо ребра от u до v, мы могли бы пойти по пути $u \to w_1 \to w_2 \to \ldots \to w_k \to v$. Это значит, что из всех рёбер между башнями высоты 2 можно оставить только те, которые ведут в ближайшую слева и справа башню, которые мы предпосчитали. Это позволяет легко посчитать для каждой вершины v высоты 2 значения l_v — количество достижимых слева, и r_v — справа, а также итоговый ответ. Теперь рассмотрим башни v высоты 1. Используя критерий из предыдущей подзадачи, мы можем понять какое множество башен высоты 1 достижимо — это может быть либо только исходная башня, либо максимальный по включению отрезок башен высоты 1, обозначим его за $[a, b]$. Нам осталось посчитать количество достижимых

башен высоты 2. Заметим, что в любом пути из v первая башня высоты больше 1 должна быть либо $a-1$-й, либо $b+1$-й. Тогда достаточно добавить к ответу l_{a-1}, если вершина $a-1$ достижима, и r_{b+1}, если вершина $b+1$ достижима. Решение работает за $O(n)$.

Сделаем наблюдение, полезное для полного решения. Мы хотим понять из каких вершин есть ребро в вершину v. Пусть gl_v — ближайший элемент в массиве слева от v, больший v, gr_v — аналогичный элемент справа. Если к некоторой вершине u есть более близкий чем v элемент, больший v-го, его можно выбрать из множества $\{gl_v, gr_v\}$. Это значит, что множество вершин, из которых проведено ребро в v — отрезок $\left[\left\lfloor\frac{gl_v+v}{2}\right\rfloor+1, \left\lceil\frac{gr_v+v}{2}\right\rceil-1\right]$.

Используя этот факт, можно получить решение пятой подзадачи. Пусть в вершину v ведут рёбра с отрезка $[c_v, d_v]$. Мы переберём v, посчитаем отрезок вершин, из которых достижима v, и добавим 1 к ответу на этом отрезке. Чтобы посчитать отрезок вершин, откуда достижима v, будем хранить текущий ответ (изначально это $[c_v, d_v]$), и отрезок рассмотренных вершин (изначально это $[v, v]$). После этого, пока отрезок рассмотренных вершин и ответ не совпадут, будем расширять отрезок рассмотренных вершин на 1 (слева или справа). Пусть u — вершина, которую мы только что добавили в отрезок рассмотренных, она должна принадлежать отрезку ответа, и мы можем объединить отрезок ответа с отрезком $[c_u, d_u]$. Когда отрезок ответа и рассмотренных вершин совпадут, не будет существовать вершин вне отрезка, из которых есть ребро в отрезок, следовательно это все вершины, из которых достижима v. Решение работает за $O(n^2)$. Это решение можно соптимизировать до решения шестой подзадачи: чтобы расширить текущий отрезок $[l, r]$, мы можем найти минимум c_v и максимум d_v на $[l, r]$ с помощью sparse table, и заменить границы на этот максимум и минимум. Можно доказать, что это решение работает за $O(n \log n)$.

Чтобы продвинуться дальше, нам понадобится ещё несколько наблюдений. Заметим, что поскольку ребро (i, j) проведено тогда, когда j-й элемент является максимумом на отрезке, где i является серединой, а j – одним из концов, из наличия ребра (i, j) следует наличие рёбер (k, j) для $i < k < j$. Аналогичное утверждение можно сделать для ребра (i, j) когда $j < i$. Пусть в некотором пути есть два соседних ребра (u, v) и (v, w), такие, что $w < u < v$. Тогда, используя свойство выше, мы получаем, что есть ребро (u, w). Делая такие замены для произвольного пути, можно добиться того, чтобы все рёбра пути вели либо только налево, либо только направо. Посчитаем кол-во достижимых по рёбрам налево из каждой вершины, аналогично по рёбрам направо, и, сложив результаты, получим ответ. Пусть L_i – самая правая вершина левее i, в которую есть ребро из i. Докажем, что можно оставить только рёбра $i \to L_i$, и количество достижимых слева не изменится. Действительно, пусть в каком-то пути мы пошли по ребру $i \to j$ и $j < L_i$. Мы знаем что тогда должно быть ребро $L_i \to j$, и $i \to j$ можно заменить на $i \to L_i$, $L_i \to j$. Делая такие замены, мы можем удалить все «плохие» рёбра. Тогда ответ для вершины i — это ответ для L_i плюс 1. Это значит, что для решения задачи достаточно уметь эффективно находить L_i для каждого i.

Мы умеем за линейное время посчитать gl_i и gr_i для всех i, используя стандратный алгоритм со стеком. Поскольку входящие рёбра в каждую вершину ведут из отрезка, который мы можем вычислить, задача о подсчёте L_i сводится к следующей: даны n отрезков $[l_i, r_i]$, для каждой позиции нужно вычислить минимальный индекс отрезка, который её покрывает. Существует множество решений этой задачи. Мы можем сделать сканлайн, который поддерживает set отрезков, которые покрывают данную позицию, тогда для каждого отрезка нужно создать событие «добавить i» в позции l_i и «удалить i» в позиции $r_i + 1$. Такое решение работает $O(n \log n)$. Также можно решать задачу с помощью дерева отрезков (тоже за $O(n \log n)$) или с помощью системы непересекающихся множеств (за $O(n \cdot \alpha(n))$).

Теперь нам нужно отвечать на запросы, используя знание о том, что можно посчитать массивы L и R, такие, что из вершины v достижимы вершины $v, L_v, L_{L_v}, \ldots, R_v, R_{R_v}, \ldots$. Мы можем представлять это как два дерева: построим «левый лес деревьев» из рёбер $v \to L_v$ и «правый лес деревьев» из рёбер $v \to R_v$. Тогда, множество достижимых из вершины v — это объединение пути до корня в левом лесу и в правом лесу.

В седьмой подзадаче есть дополнительное ограничение на запросы $u_i = v_i$. Мы можем разделить отрезок запроса на две части: то, что левее u и то, что правее u, и решать задачу отдельно для каждой из частей. Чтобы ответить на запрос про левую часть, посчитаем двоичные подъёмы на левом лесу. Чтобы найти количество вершин на пересечении пути до корня и отрезка $[l, r]$, найдём

двоичными подъёмами первую вершину на пути, не большую r, и первую, строго меньшую l (по индексу). Тогда ответ — это разность глубин этих вершин. Решение работает за $O(n \log n)$.

В восьмой подзадаче дополнительное ограничение $r \leqslant \min(u, v)$ или $l \geqslant \max(u, v)$. Будем считать, что $l \leqslant r \leqslant u \leqslant v$, остальные случаи аналогичны. Заметим, что на отрезке $[l, r]$ нет вершин, которые находятся правее u или v, поэтому мы можем забыть про существование правого леса. В левом лесу пересечение множества достижимых — это путь от $\mathrm{lca}(u, v)$ до корня. Чтобы ответить на запрос, найдём LCA, а затем посчитаем пересечение пути до корня и отрезка аналогично предыдущей подзадаче. Решение работает за $O(n \log n)$.

В девятой подзадаче отрезок каждого запроса содержит все позиции. Пусть $u < v$. Мы разделим его на три части: левее u, между u и v, и правее v. Для первой и третьей части мы уже умеем считать ответ как в прошлой подзадаче. Чтобы посчитать количество общих достижимых между u и v, понадобится наблюдение:

Пусть есть три вершины $i \leqslant j < k$, и $a_k < a_j$. Тогда k не достижима из i.

Доказательство: докажем это по индукции по убыванию j. Пусть для какого-то j путь из i до k нашёлся. Рассмотрим первую вершину w в этом пути, большую j. Заметим, что поскольку мы пришли в w из вершины не правее j, j-я позиция будет к этой вершине ближе, чем w-я, так что должно выполняться неравенство $a_w \geqslant a_j > a_k$. Применяя предположение индукции для $j := w$, получаем, что пути от i до k не существует, противоречие.

Из этого следует, что элементы пересечения множеств достижимых из u и из v в середине являются максимумами на отрезке $[u, v]$. Действительно, если это не так, пусть w достижима из u и из v, и на $[u, v]$ есть вершина s, такая что $a_s > a_w$. Пусть $s < w$, тогда w не может быть достижима из u по предыдущему утверждению, противоречие. Случай $s > w$ аналогичен.

Рассмотрим множество максимумов на отрезке $[u, v]$. Заметим, что подмножество, состоящее из максимумов, достижимых из u – это подотрезок пути из u в правом лесу. Это также подотрезок в массиве индексов максимумов, поскольку ребро не может переходить «через максимум». С помощью RMQ найдём максимум, с помощью двоичных подъёмов по лесам найдём отрезки достижимых максимумов, пересечём их, и вычислим количество элементов как разность глубин. Решение работает за $O(n \log n)$.

Десятая подзадача не сильно сложнее. Теперь у нас есть нетривиальные отрезки, мы будем делить их на три части аналогично девятой подзадаче. Обработку левой и правой части можно сделать как в восьмой подзадаче, подсчёт ответа для части посередине такой же, но нужно пересечь ответ с отрезком запроса (например, используя двоичные подъёмы). Решение по-прежнему работает за $O(n \log n)$.

BONUS: на самом деле при решении задачи для отрезка между u и v можно не использовать факт про максимумы и решать более общую версию задачи:

Даны два корневых дерева на вершинах $1, 2, \ldots, n$, нужно уметь быстро пересекать два пути до корня, исходящие из вершин запроса u и v, и выводить количество вершин в пересечении.

Это можно делать следующим образом: запустим dfs по первому дереву, поддерживая массив w, в котором в позиции i находится 1 тогда и только тогда, когда вершина i является предком текущей вершины в dfs и 0 иначе. При обходе дерева у нас происходят точечные изменения в этом массиве. Когда мы находимся в вершине u, мы можем ответить на все запросы вида (u, v): достаточно взять сумму на вертикальном пути второго дерева по текущему состоянию массива w. Мы свели задачу к задаче на дереве, где в вершинах меняются веса и нужно искать сумму на пути. Эта задача эквивалента задаче, где значения прибавляются ко всем вершинам в поддереве и запрос — найти значение в вершине. Это можно решать с помощью дерева отрезков на эйлеровом обходе. С учётом того, что во втором дереве все рёбра ведут направо, это решение можно адаптировать для последней подзадачи, где отрезок — это не весь массив.