Задача A. Идеальный бутерброд

Посчитаем количество бутербродов, которое съест каждый из гостей. Нетрудно убедиться, что это количество равно $\lfloor \frac{N}{M} \rfloor$. Тогда всего будет съедено $M \cdot \lfloor \frac{N}{M} \rfloor$ бутербродов, а останутся несъеденными $N - M \cdot \lfloor \frac{N}{M} \rfloor$. Это количество равно остатку от деления N на M.

Далее необходимо умножить количество съеденных на 2, количество несъеденных на 80 и сложить эти две величины.

Задача B. Очередная головоломка

Задачу можно решить двумя способами: либо перебрать все шестизначные числа и найти такое, которое удовлетворяет всем условиям, либо аналитически. Разберём ниже второй способ.

Поскольку все цифры различны, $\overline{ef} \mid \overline{ab}$ и $\overline{ef} \mid \overline{cd}$, то $3 \cdot \overline{ef} < 100$, так как нам нужны хотя бы 3 различных двузначных числа, которые делятся на $\overline{ef}$.

Пусть $\overline{ef}$ нечётное, тогда $\overline{ab}$ и $\overline{cd}$ также нечётные (так как цифры $\overline{b}$, $\overline{d}$ и $\overline{f}$ нечётные). Среди таких нам подходят только 25, 27 и 29 (у остальных либо ведущий ноль, либо первая и вторая цифры нечётные, либо разница между ними $\leqslant 1$, либо $3 \cdot \overline{ef} \geqslant 100$). Это означает, что $\overline{ab}$ и $\overline{cd}$ могут быть только $2 \cdot \overline{ef}$ или $3 \cdot \overline{ef}$, однако $2 \cdot \overline{ef}$ чётное, что противоречит предположению.

Пусть $\overline{ef}$ чётное, тогда $\overline{ab}$ и $\overline{cd}$ также чётные (так как цифры $\overline{b}$, $\overline{d}$ и $\overline{f}$ чётные). Среди таких нам подходят только 14, 16, 18 и 30 (у остальных либо ведущий ноль, либо первая и вторая цифры чётные, либо разница между ними $\leqslant 1$, либо $3 \cdot \overline{ef} \geqslant 100$). 30 можно сразу же отбросить, так как и у 30, и у 60, и у 90 в конце нули. Рассмотрим все остальные варианты:

- $\overline{ef} = 14$ — рассмотрим все двузначные числа, которые делятся на 14: 14 подходит; 28 не подходит, обе цифры чётные; 42 не подходит, обе цифры чётные; 56 не подходит, разница между цифрами 1; 70 подходит; 84 не подходит, обе цифры чётные; 98 не подходит, разница между цифрами 1. Подходят только два варианта, чего не хватает, значит $\overline{ef}$ не 14.

- $\overline{ef} = 16$ — рассмотрим все двузначные числа, которые делятся на 16: 16 подходит; 32 не подходит, разница между цифрами 1; 48 не подходит, обе цифры чётные; 64 не подходит, обе цифры чётные; 80 не подходит, обе цифры чётные; 96 подходит. Подходят только два варианта, чего не хватает, значит $\overline{ef}$ не 16.

- $\overline{ef} = 18$ — рассмотрим все двузначные числа, которые делятся на 18: 18 подходит; 36 подходит; 54 не подходит, разница между цифрами 1; 72 подходит; 90 подходит. Подходят четыре варианта, поймём, как строить дальше.

  $\overline{abcdef} = \overline{abcd18}$, значит, что $\overline{d}$ не может быть равно 0 или 2, значит $\overline{cd} = 36$.

  $\overline{abcdef} = \overline{ab3618}$, значит, что $\overline{b}$ не может быть равно 2, значит $\overline{ab} = 90$.

В итоге есть единственный ответ: $\overline{abcdef} = 903618$.

Задача C. Идеальное очищение

Найдём pos_O и pos_S — позиции фигур `O` и `S` в последовательности.

- Если $pos_O < pos_S$, то фигура `O` появляется раньше `S`, поэтому можно использовать `SINGLE DOUBLE PERFECT CLEAR`.

- Иначе $pos_O > pos_S$, то есть фигура `S` появляется раньше `O`, поэтому можно использовать `TRIPLE TRIPLE PERFECT CLEAR`.

Задача D. Финальный босс

Герою с книгой достаточно один раз телепортироваться в локацию с боссом, до этого собрав множество героев. Пусть он должен собрать m героев, включая себя. Тогда он за $m - 1$ секунд заберёт всех остальных $m-1$ героев и за 1 секунду телепортируется в локацию с боссом. Получается, все m героев прибудут в локацию с боссом за m секунд.

Остальные герои будут добираться самостоятельно, причём если мы сказали герою добираться самому за d_i, то любой герой j с $d_j \leqslant d_i$ тоже может добраться самостоятельно, время сбора от этого не увеличится. Отсортируем массив d, тогда можно считать, что самостоятельно добирающиеся герои образуют префикс, а телепортирующиеся суффикс. Переберём количество телепортирующихся героев m, тогда время сбора равно $\max(m, d_{n-m})$ для $m < n$. Если $m = n$, то мы забираем всех героев за n.

Задача E. Игра любителей

Для начала рассмотрим игру без зелёных кучек. Если у игроков не равное количество крышечек своего цвета, то побеждает игрок, у которого больше крышечек, для этого ему достаточно каждый ход брать по одной крышке. Если же количество равно, то выигрывает второй игрок, пользуясь такой же стратегией.

Теперь вернем в игру зелёные кучки. Если у одного из игроков больше крышек своего цвета, чем у другого, то у него есть выигрышная стратегия — он может брать сколько угодно зелёных крышечек, пока они есть, после чего игра сведется к игре без зелёных кучек, которая разбиралась выше.

Если же у игроков поровну крышек своего цвета, то игра сводится к игре в обычный ним на зелёных кучках, потому что если кто-то возьмет камень своего цвета, то игра сведется к предыдущему случаю и взявший проиграет, соответственно игроки хотят сыграть на зелёных кучках так, что после того как они закончатся, ходить будет их оппонент, что эквивалентно игре в классический ним, в котором проигрывает игрок, который не может взять камень из какой-либо кучки.

Чтобы узнать победителя при оптимальной игре в ним, достаточно посчитать побитовый xor(исключающее или) всех размеров кучек, и если он равен 0, то побеждает второй игрок, иначе побеждает первый.

Задача F. Сказания о любителях

Пусть S_g — множество обычных сундуков, а S_s — множество сундуков котегов. Если $c_g < \min_{S_g}(x_i)$ и $c_s < \min_{S_s}(x_i)$, то нельзя открыть ни одного сундука и ответ «`Still Bored`». Если же $c_g \geqslant min_{S_g}(x_i)$ и $c_s \geqslant min_{S_s}(x_i)$, то мы можем изначально открыть два сундука разных типов. Остаётся вариант, когда мы можем открыть некоторые сундуки одного типа и не можем открыть ни одного сундука второго. Без ограничения общности, пусть мы можем открыть какие-то обычные сундуки. Тогда наша задача сводится к следующей: необходимо выбрать S, являющееся подмножеством S_g, такое, что $\sum_S(x_i) \leqslant c_g$ и $c_s + \sum_S(y_i) \geqslant \min_{S_s}(x_i)$. Заметим, что нам достаточно найти S', являющееся подмножеством S_g, такое что $\sum_{S'}(x_i) \leqslant c_g$ и $\sum_{S'}(y_i)$ — максимально, после чего проверить только это множество и вывести ответ. Поиск множества S' является классической задачей о рюкзаке, которая решается за $O(N \cdot c_g)$ по времени и памяти.

Задача G. Правильная сушилка

Баланс скобочной последовательности — это количество открывающих скобок `'('` минус количество закрывающих скобок `')'`. Последовательность скобок правильная тогда и только тогда, когда баланс каждого её префикса неотрицательный и баланс всей скобочной последовательности равен нулю.

Посчитаем величины: a_i — баланс на префиксе длины i, если все вопросы заменить на `')'`, b_i — баланс на префиксе длины i, если все вопросы заменить на `'('`, $c_i = \min_{0 \leqslant j \leqslant i} a_j$ и $d_i = \min_{0 \leqslant j \leqslant i} b_j$. Для префикса длины k, чтобы итоговый баланс был равен нулю, нужно ровно $\frac{-a_k}{2}$ вопросов заменить на `'('`, а остальные на `')'`, поэтому a_k должно быть чётным и неположительным. Вопросов меньше, чем $\frac{-a_k}{2}$, тогда и только тогда, когда $b_k < 0$, то есть даже при замене всех вопросов на `'('` итоговый баланс отрицательный, получается, нужно проверить $b_k \geqslant 0$. Если мы как-то заменили вопросы на скобки, то мы можем двигать открывающие скобки влево, это только увеличивает балансы на префиксах, то есть оптимальнее всего заменить первые $\frac{-a_k}{2}$ вопросов на `'('`.

Пусть j — позиция $\frac{-a_k}{2}$-го вопроса (если $\frac{-a_k}{2} = 0$, то $j = 0$), рассмотрим балансы на префиксах:

- Если $h \leqslant j$, то на префиксе длины h все вопросы заменены на открывающие скобки, баланс префикса равен b_h, а минимальный баланс среди таких префиксов равен d_j, то есть нам до-

статочно проверить $d_j \geqslant 0$. Покажем, что также выполняется $a_h \geqslant a_k$. У нас верно $b_h \geqslant 0$, но $b_h \leqslant a_h + 2 \cdot \frac{-a_k}{2}$, ведь вопросов на префиксе h не больше $\frac{-a_k}{2}$, следовательно, $a_h + 2 \cdot \frac{-a_k}{2} \geqslant 0$ для $0 \leqslant h \leqslant j$, что то же самое, что и $a_h \geqslant a_k$.

- Для префиксов длины $h > j$ баланс равен $a_h + 2 \cdot \frac{-a_k}{2} = a_h - a_k$, то есть должно выполняться $a_h \geqslant a_k$ для $j < h \leqslant k$. Мы доказали в первом случае, что $a_h \geqslant a_k$ для $h \leqslant j$, а значит, если объединить утверждения, то $a_h \geqslant a_k$ для $0 \leqslant h \leqslant k$ тоже верно, что равносильно $a_k = c_k$.

Причём условие $d_j \geqslant 0$ можно проверить, не зная позицию j, потому что для $h > j$ мы проверили неравенство $a_h + 2 \cdot \frac{-a_k}{2} \geqslant 0$. У нас $b_h \geqslant a_h + 2 \cdot \frac{-a_k}{2}$ для $h > j$, потому что на префиксе длины $h > j$ уже встретилось не меньше $\frac{-a_k}{2}$ вопросов, а значит, $b_h \geqslant 0$ уже верно. Можно вместо $d_j \geqslant 0$ проверять $d_k \geqslant 0$. Получается, ответ «`YES`» тогда и только тогда, когда a_k чётно, $a_k \leqslant 0$, $c_k = a_k$ и $d_k \geqslant 0$.

Задача H. Пазлы для любителя

Для начала найдем кусочек, у которого есть две соседние стороны, являющиеся граничными (имеют значение 0), поставим его в верхний левый угол и повернем так, чтобы верхняя и левая стороны были граничными (имели значение 0), после чего сможем восстановить остальной пазл построчно. Заметим, что если строка еще не закончена, то правая сторона самого правого кусочка будет не граничной (иметь значение не 0), а так как все типы соединений между деталями уникальны, то существует ровно один элемент, который можно поставить справа от самого правого элемента в строке, то же самое истинно и для столбцов. Из этих замечаний можно придумать следующий алгоритм: пока можем поставить элемент после самого правого в самой нижней строке, ставим его, после чего ставим, если можем, новый элемент под самым левым из самых нижних, создавая тем самым новую строку. Искать подходящие элементы можно за $O(1)$ с помощью массива, в котором для каждого типа стороны указан номер элемента с такой стороной.

Задача I. Двоичная сортировка 2

Если a, b — биты, то $\min(a, b) = a \& b$ и $\max(a, b) = a|b$.

Считаем все Q запросов заранее, для каждого конвейера сохраним битсет длины Q, содержащий биты всех запросов на этот конвейер. Тогда для перемычки между соседними конвейерами с битсетами c_i и c_{i+1} после сортировки значения станут $c_i \& c_{i+1}$ и $c_i | c_{i+1}$, перемычку можно так обрабатывать за $\frac{Q}{64}$. Всего перемычек не больше $\frac{N \cdot M}{2}$, поэтому операций получится около $\frac{N \cdot M \cdot Q}{64}$.

Задача J. Подземелья и любители

Заметим, что в задаче нас просят посчитать количество обходов поиска в глубину, лексикографически меньших данной последовательности. Пусть мы начали обход в вершине s, подвесим дерево за вершину s. Заметим, что, зайдя в какую-либо вершину, мы обязательно пойдем из нее во всех ее детей в каком-то порядке, причем на это не влияют ни вершины, посещенные раньше, ни вершины, которые будут посещены позже. Из этого мы можем легко вывести, что число обходов поиска в глубину, стартующих из вершины s — $deg(s) \cdot \prod_{i=1}^{N} (deg(i) - 1)!$. Теперь предположим, что мы запустили наш обход поиска в глубину и уже посетили несколько вершин, тогда посчитать количество обходов, имеющих такой префикс, можно по формуле $\prod_{i=1}^{N}$(количество непосещёных детей вершины i)!

Очевидно, что любой обход, начатый из вершины $< p_0$, будет лексикографически меньше p, а любой обход, начатый из вершины $> p_0$, будет лексикографически больше. Заведем дополнительную переменную $cnt = \prod_{i=1}^{N} (deg(i) - 1)!$, с помощью которой сможем легко пересчитать количество обходов из всех вершин $< p_0$ и сложить их. Теперь осталось посчитать обходы, начинающиеся в вершине p_0. Теперь у нас есть какой-то префикс последовательности длины l, с которым мы совпадаем, и вершина, в которой мы сейчас находимся. Если у вершины нет детей, которых мы еще не посещали в обходе, то надо вернуться в предыдущую вершину, если же непосещённые дети есть, то рассуждения идентичны предыдущим: пойдя в вершину $< p_l$, обход гарантированно будет $< p_l$, а, пойдя в вершину $> p_l$, будет гарантированно больше. Чтобы просуммировать количество обходов, идущих в вершину $< p_l$, необходимо знать количество таких вершин, а также количество способов продолжить обход. Формула для количества способов нам уже известна, и мы можем ее

динамически поддерживать в процессе посещения новых вершин, для этого достаточно делить ее на количество непосещенных детей вершины, из которой мы сейчас куда-то пойдем. Для подсчета количества вершин $< p_l$, в которые мы можем пойти из данной, нам необходима структура, которая умеет удалять элемент и искать количество меньших; структура данных *orderedset* позволяет делать обе эти операции за $O(log(N))$, и тем самым мы получаем итоговую асимптотику $O(N \cdot log(N))$

Задача K. Прогрессивность

Поймём, что такое прогрессивность. Это длина наибольшего по размеру строго возрастающего подотрезка, содержащего этот элемент. Пусть эта длина равна x. Тогда для каждого элемента из этого подотрезка прогрессивность равна x, а сумма прогрессивностей на этом подотрезке равна x^2. Тогда, чтобы посчитать суммарную прогрессивность, надо разбить весь массив на максимальные по включению подотрезки, где значения строго возрастают, и посчитать сумму квадратов их длин. Нетрудно заметить, что такое разбиение единственное (можно показать, пройдя жадно).

Будем решать эту задачу с помощью дерева отрезков. В вершине будем хранить следующие величины:

- длина отрезка (sz);
- значение левого элемента отрезка (l);
- значение правого элемента отрезка (r);
- длина наибольшего по включению возрастающего подотрезка этого отрезка, содержащего левый элемент отрезка (ls);
- длина наибольшего по включению возрастающего подотрезка этого отрезка, содержащего правый элемент отрезка (rs);
- суммарная прогрессивность для этого отрезка (ans).

Чтобы доказать, что дерево отрезков работает, нужно показать несколько вещей:

1. Для отрезков длины 1 все 6 величин считаются очевидно.
2. Как найти величины для отрезка u, зная величины для его детей v и w (v — левый, w — правый)?
   - $u_l = v_l$
   - $u_r = w_r$
   - $u_{sz} = v_{sz} + w_{sz}$
   - если $v_{ls} = v_{sz}$ и $v_r < w_l$, то $u_{ls} = v_{ls} + w_{ls}$, в противном случае $u_{ls} = v_{ls}$
   - если $w_{rs} = w_{sz}$ и $v_r < w_l$, то $u_{rs} = v_{rs} + w_{rs}$, в противном случае $u_{rs} = w_{rs}$
   - если $v_r < w_l$, то $u_{ans} = v_{ans} + w_{ans} - v_{rs}^2 - w_{ls}^2 + (v_{rs} + wls)^2 = v_{ans} + w_{ans} + 2 \cdot v_{rs} \cdot w_{ls}$, в противном случае $u_{ans} = v_{ans} + w_{ans}$
3. Что происходит с отрезком v, когда к каждому элементу прибавляют одну и ту же величину x?
   - $v_l \mathrel{+}= x$
   - $v_r \mathrel{+}= x$
   - нетрудно заметить, что, поскольку разности соседних элементов остались такими же, то v_{sz}, v_{ls}, v_{rs} и v_{ans} не изменились
4. Что происходит с отрезком v, когда каждому элементу присваивается одно и то же значение x?

- $v_l = x$
- $v_r = x$
- $v_{ls} = 1$, так как следующий элемент лишь равен ему
- $v_{rs} = 1$, так как предыдущий элемент лишь равен ему
- $v_{ans} = v_{sz}$, так как каждый элемент равен соседям, длина каждого отрезка разбиения равна 1
- v_{sz} не изменяется

Детали реализации дерева отрезков с прибавлениями на подотрезке и с присвоениями на подотрезке мы здесь приводить не будем.

Задача L. የእሃዞችድሞር

Назовём $sum(x)$ — сумму цифр x в B-ичной системе счисления, $xor(x)$ — побитовое исключающее «ИЛИ» цифр x в B-ичной системе счисления, $f(x) = sum(x) + xor(x)$.

Утверждение. Если $x \geqslant 2 \cdot B$, то $f(x) < x$.

Доказательство. Поскольку $xor(x) \leqslant sum(x)$, то $f(x) \leqslant 2 \cdot sum(x)$. Пусть у числа x ровно n цифр. Тогда $x = d_0 \cdot B^0 + d_1 \cdot B^1 + \ldots + d_{n-1} \cdot B^{n-1}$, а $sum(x) = d_0 + \ldots + d_{n-1}$. Заметим, что для любого $i > 0$ верно $d_i \cdot B^i \geqslant 2 \cdot d_i$, так как $B \geqslant 3$. При этом $d_0 \leqslant B - 1$. Рассмотрим $x - 2 \cdot sum(x)$:

$$d_0 \cdot B^0 + d_1 \cdot B^1 + \ldots + d_{n-1} \cdot B^{n-1} - 2 \cdot (d_0 + \ldots + d_{n-1}) = -d_0 + \sum_{i=1}^{n-1} d_i \cdot (B^i - 2)$$

Разберём два случая:

- $n \geqslant 3$. Тогда $\sum_{i=1}^{n-1} d_i \cdot (B^i - 2) \geqslant B^2 - 2$. Значит

  $$x - f(x) \geqslant x - 2 \cdot sum(x) \geqslant B^2 - 2 - (B - 1)$$

  что больше нуля для любого $B \geqslant 3$.

- $n = 2$. Тогда $d_1 \geqslant 2$, так как $x \geqslant 2 \cdot B$. Значит

  $$x - f(x) \geqslant x - 2 \cdot sum(x) = d_1 \cdot (B-2) - d_0 \geqslant d_1 \cdot (B-2) - (B-1) = (d_1 - 1) \cdot (B-2) - 1 \geqslant (B-2) - 1 = B - 3$$

  то есть, если $B > 3$, то $x - f(x) > 0$. Если же $B = 3$, то можно рассмотреть все числа $\geqslant 6$ отдельно:

  - ✓ $f(6) = f(20_3) = 2 + 2 = 4 < 6$
  - ✓ $f(7) = f(21_3) = 3 + 3 = 6 < 7$
  - ✓ $f(8) = f(22_3) = 4 + 0 = 4 < 8$

Исходя из этого утверждения, x будет уменьшаться, пока не станет $< 2 \cdot B$. Заметим, что сумма цифр и побитовое исключающее «ИЛИ» цифр любого числа имеют одинаковую чётность, поэтому $f(x)$ всегда чётные. Это означает, что все возможные циклы состоят только из чётных чисел $< 2 \cdot B$. Будем рассматривать только такие числа. Если x однозначное число, то $f(x) = 2 \cdot x$. Если x двузначное число и $x = B + r$, то $f(x) = (r + 1) + (r \oplus 1)$. Здесь есть два случая:

1. Если B нечётное, то r также нечётное (так как x чётное), поэтому $r \oplus 1 = r - 1$, а это значит, что $f(x) = 2 \cdot r$. Иными словами, при нечётном B функция $f(x) = 2 \cdot (x \bmod B)$. Все возможные значения этой функции различны по модулю B, а это значит, что если ответ на задачу равен k, то $x \equiv x \cdot 2^k \mod B$, при этом k минимальное. Нетрудно убедиться, что если мы найдём такое k, то это и будет ответом на задачу.

2. Если B чётное, то r также чётное (так как x чётное), поэтому $r \oplus 1 = r + 1$, а это значит, что $f(x) = 2 \cdot r + 2 = 2 \cdot (x - B) + 2 = 2 \cdot x - 2 \cdot (B - 1)$. То есть:

- Для $x < B \quad f(x) = 2 \cdot x$
- Для $x \geqslant B \quad f(x) = 2 \cdot x - 2 \cdot (B-1) \equiv 2 \cdot x \mod (B-1)$

Так как все значения x чётные и меньше $2 \cdot B$, то их ровно $B-1$, при этом они все дают разные остатки при делении на $B-1$. Это значит, что если ответ на задачу равен k, то $x \equiv x \cdot 2^k \mod (B-1)$, при этом k минимальное. Нетрудно убедиться, что если мы найдём такое k, то это и будет ответом на задачу.

То есть, ответом на задачу в первом случае является такое минимальное k, что $2^k \equiv 1 \mod B$, а во втором случае — такое минимальное k, что $2^k \equiv 1 \mod (B-1)$. Расскажем, как решать первый случай, второй решается по аналогии.

По теореме Эйлера $2^{\phi(B)} \equiv 1 \mod B$ (так как 2 и нечётное B взаимно просты). Подходящее k должно делить $\phi(B)$ нацело (иначе $\phi(B) = a \cdot k + r, \quad 2^{\phi(B)} = 2^{k \cdot a} \cdot 2^r \equiv 2^r \mod B$, что не может быть сравнимо с 1 по определению k). Переберём все делители $\phi(B)$, возведём 2 в их степень по модулю и сравним с единицей, наименьший подходящий — ответ. Используя двоичное возведение в степень, можно добиться асимптотики $O(\sqrt{\phi(n)} \cdot \log n) \leqslant O(\sqrt{n} \cdot \log n)$.

Задача M. Логистические маршруты и маршрутная логистика

Очевидно, что действия происходят на дереве. Подвесим его за произвольную вершину, например, за первую. Запустим поиск в глубину и посчитаем для каждой вершины время входа tin_v и время выхода $tout_v$, а также глубину d_v. Рассмотрим каждую логистическую сеть отдельно. Назовём заключение/расторжение контрактов добавлением и удалением вершин из множества S, а логистическую сеть для этого множества вершин назовём $STH(S)$.

Представим себе циклический бинарный массив размера N, где 1 — означает, что вершина в множестве, а 0 — нет, при этом вершины будут идти в порядке возрастания tin (то есть вершины с наименьшим и с наибольшим tin будут находиться рядом в циклическом массиве). Тогда добавление/удаление — просто смена 0 на 1 или 1 на 0. Пусть $prev1(v)$ — ближайшая слева единица в бинарном массиве, а $next1(v)$ — ближайшая справа (если в бинарном массиве v единица, то $prev1(v) = next1(v) = v$).

Назовём v_i вершину с номером i, если все вершины S отсортировать в порядке возрастания tin. Поймём, какие вершины лежат в $STH(S)$. Это все вершины, которые лежат на путях от $LCA(v_1, v_2, \ldots, v_{|S|})$ до каждой из v_i. Очевидно, что это дерево с корнем $root = LCA(v_1, v_2, \ldots, v_{|S|})$.

Лемма 1. $root = LCA(v_1, v_{|S|})$.

Доказательство. По определению LCA $root$ — это самая низкая вершина u, для которой верно $tin_u \leqslant tin_{v_i}$ и $tout_u \geqslant tout_{v_i}$ для каждого i. Рассмотрим вершину v_m с наибольшим $tout$. Очевидно, что $root = LCA(v_1, v_m)$. Есть три случая, чем может быть v_m:

1. $v_m = v_1$, тогда $root = v_1 = LCA(v_1, v_{|S|})$.
2. $v_m = v_{|S|}$, тогда $root = LCA(v_1, v_{|S|})$.
3. $tin_{v_m} < tin_{v_{|S|}}$ и $tout_{v_m} > tout_{v_{|S|}}$, тогда v_m — предок $v_{|S|}$. Это означает, что v_m лежит на пути от $root$ до $v_{|S|}$, также на этом пути лежит вершина $w = LCA(v_1, v_{|S|})$. Если w — предок v_m или $w = v_m$, то, поскольку w не может быть предком $root$ (по определению $root$), $w = LCA(v_1, v_m) = root$; в противном случае v_m предок w, однако это означает, что $tin_{v_m} < tin_w \leqslant tin_{v_1}$, что не может быть правдой по определению v_1.

Лемма 2. Вершина v лежит в STH тогда и только тогда, когда она лежит на пути между $prev1(v)$ и $next1(v)$.

Доказательство. Если вершина на пути между $prev1(v)$ и $next1(v)$, то она принадлежит STH по определению. Теперь покажем, что если вершина в STH, то она находится на пути между $prev1(v)$ и $next1(v)$. Если $v = prev1(v) = next1(v)$, то вершина точно на пути. Осталось рассмотреть два случая:

1. $tin_{prev1(v)} < tin_v < tin_{next1(v)}$. Поскольку v лежит в STH, то лежит на каком-то пути от v_x до v_y, а это означает, что v является предком одной из них, без ограничения общности, пусть v_x, тогда $tin_v < tin_{v_x}$. Это означает, что $next1(v)$ должна лежать в поддереве v (по определению $next1(v)$). Как известно, путь из $prev1(v)$ в $next1(v)$ проходит через $LCA(prev1(v), next1(v))$, который, как и v, является предком $next1(v)$. Рассмотрим два варианта:

   - $LCA(prev1(v), next1(v))$ является предком v, тогда путь от $next1(v)$ до $LCA(prev1(v), next1(v))$ должен сначала подняться до v, после чего подняться от v до $LCA(prev1(v), next1(v))$, следовательно v лежит на пути от $prev1(v)$ до $next1(v)$;
   - $LCA(prev1(v), next1(v)) = v$ или v является предком $LCA(prev1(v), next1(v))$, но тогда $tin_v \geqslant tin_{LCA(prev1(v),next1(v))} \geqslant tin_{prev1(v)}$, что противоречит определению $prev1(v)$, следовательно такой случай невозможен.

2. $prev1(v) = v_{|S|}$ и $next1(v) = v_1$. Рассмотрим два варианта:

   - $tin_v < tin_{v_1}$. Покажем, что $tout_v > tout_{v_1}$. Пусть это не так. Поскольку v лежит в STH, то лежит на каком-то пути от v_x до v_y, а это означает, что v является предком одной из них, без ограничения общности, пусть v_x, но тогда $tin_v < tin_{v_x} < tin_{v_1}$, противоречие с определением v_1. Значит, v — предок v_1, а значит, лежит на пути от $root$ до v_1, а значит и на пути от v_1 до $v_{|S|}$.
   - $tin_{v_{|S|}} < tin_v$. Поскольку v лежит в STH, то лежит на каком-то пути от v_x до v_y, а это означает, что v является предком одной из них, без ограничения общности, пусть v_x, но тогда $tin_{v_{|S|}} < tin_v < tin_{v_x}$, что противоречит определению $v_{|S|}$, значит, такой случай невозможен.

С помощью леммы 2 мы можем проверить, лежит ли вершина v в STH, найдя $prev1(v)$ и $next1(v)$ и проверив, что v лежит на пути между ними. Для этого нужно проверить два пути от $LCA(prev1(v), next1(v))$ до $prev1(v)$ и до $next1(v)$ (делается легко с помощью tin и $tout$).

Теперь решим другую задачу. Пусть дана вершина v, нужно найти минимальное расстояние между v и вершиной из $STH(S)$. Назовём вершину из $STH(S)$, в которой достигается минимум, $closest(v)$.

Лемма 3. $closest(v)$ — самая глубокая из $LCA(prev1(v), v)$, $LCA(v, next1(v))$ и $LCA(prev1(v), next1(v))$.

Доказательство. Если v лежит в STH, по лемме 2 она лежит на пути между $prev1(v)$ и $next1(v)$, и лемма 3 очевидно верна. Пусть это не так. Следуя аналогичным лемме 2 рассуждениям, нетрудно убедиться, что $closest(v)$ лежит на пути между $prev1(v)$ и $next1(v)$. Пусть вершина $w = LCA(prev1(v), next1(v))$. Если v не лежит в поддереве w, то $closest(v) = w$, и самая глубокая из $LCA(prev1(v), v)$, $LCA(v, next1(v))$ и w это действительно w. Если же v лежит в поддереве w, то $closest(v)$ лежит на одном из путей от w до $prev1(v)$ и до $next1(v)$. Легко заметить, что в первом случае $closest(v) = LCA(prev1(v), v)$ и $LCA(v, next1(v)) = w$, а во втором случае $closest(v) = LCA(v, next1(v))$ и $LCA(prev1(v), v) = w$. В обоих случаях $closest(v)$ глубже w или совпадает с ней.

С помощью $closest(v)$ и глубин каждой из вершин легко найти расстояние от v до STH — $d_v + d_{closest(v)} - 2 \cdot d_{LCA(v,closest(v))}$.

Наконец, перейдём к решению общей задачи. Для этого нам пригодятся две STH. Расстояние между ними — минимум из расстояний от вершин одной из них до другой. Если это расстояние равно 0, то существует вершина, которая принадлежит как STH_1, так и STH_2, другими словами, они пересекаются.

Лемма 4. STH_1 и STH_2 пересекаются тогда и только тогда, когда корень одной из них лежит в другой.

Доказательство. Выберем вершину v из пересечения. Тогда $root_1$ является предком v (или $root_1 = v$) и $root_2$ является предком v (или $root_2 = v$). Это значит, что все вершины между $root_1$ и v лежат в STH_1 и все вершины между $root_2$ и v лежат в STH_2. Поскольку и $root_1$, и $root_2$ являются

предками v, то один из них лежит на пути от v к другому, а значит, и лежит в соответствующей STH.

С помощью этой леммы мы можем быстро проверить STH_1 и STH_2 на пересечение. Пусть они не пересекаются, тогда есть два случая:

1. Корень одной из STH является предком корня другой. Без ограничения общности, пусть $root_1$ — предок $root_2$, тогда все пути от вершин STH_2 к любой вершине STH_1 будут проходить через $root_2$ и будут вести к одной и той же $closest_1(root_2)$. Поэтому ответом будет расстояние между $root_2$ и $closest_1(root_2)$.

2. Ни один из корней STH не является предком другого. Это означает, что поддеревья $root_1$ и $root_2$ не пересекаются, и любой путь от вершины STH_1 к вершине STH_2 вначале поднимется до $root_1$, дойдёт до $root_2$, а затем спустится до нужной. Таким образом, ответом будет расстояние между $root_1$ и $root_2$. Нетрудно убедиться, что в этом случае $root_1 = closest_1(root_2)$ и $root_2 = closest_2(root_1)$.

Все эти случаи в задаче можно объединить в простую формулу: пусть $x = 1$, если $root_1$ глубже $root_2$, и $x = 2$ в противном случае, тогда ответом на задачу будет расстояние между $root_x$ и $closest_{3-x}(root_x)$.

Теперь поймём, что лучше всего использовать для подсчёта LCA, $prev1$ и $next1$, а также для смены 0 на 1 или 1 на 0. Для LCA можно использовать разреженную таблицу с предподсчётом за $O(N \cdot \log N)$ и ответом на запрос за $O(1)$. Для $prev1$, $next1$ и для смены 0 на 1 или 1 на 0 можно использовать дерево Фенвика, дерево отрезков, где каждая из операций выполняется за $O(\log N)$; либо не использовать бинарный массив вовсе, заменив всё на упорядоченные по tin множества S, где можно добавлять, удалять и находить следующий и предыдущий элемент по tin, каждая из операций выполняется за $O(\log N)$. Общая сложность решения равна $O((N + Q) \cdot \log N)$.

P.S. У жюри есть решение с использованием b-дерева на 64-битных масках и $b = 64$, где каждый бит вершины равен 1, если в соответствующем ребёнке есть единица, и равен 0, если нет. С использованием встроенных команд по поиску первого и последнего единичного бита в 64-битном числе можно ускорить выполнение $prev1$, $next1$ и смену 0 на 1 или 1 на 0 до $O(\log_{64} N)$.

P.P.S. Описанная в разборе STH расшифровывается как SubTree Hull (оболочка поддерева). См. задачу на timus.online «2119. Древесная оболочка».

Задача N. Растаманы

У этой задачи есть множество решений, мы опишем одно из них.

Заметим, что растаманов можно заменить на точки с целыми координатами от 1 до N, а условие — на то, чтобы никакие 3 точки не находились на одной прямой.

Вначале научимся расставлять $P-1$ растаманов на доске размера $P \times P$, где P — простое число. Для каждого i из чисел от 1 до $P-1$ поставим растамана в клетку $(i; \frac{-1}{i} \pmod P)$. Покажем, что никакие три точки не лежат на одной прямой. Очевидно, что на каждой вертикали лежит ровно одна точка. Рассмотрим все оставшиеся прямые вида $y = k \cdot x + b$. Домножим обе стороны на x (ведь у нас все координаты положительные), получится $k \cdot x^2 + b \cdot x = x \cdot y$. Очевидно, что точка $(x_0; \frac{-1}{x_0})$ лежит на этой прямой только если $k \cdot x_0^2 + b \cdot x_0 \equiv -1 \pmod P$, это квадратное уравнение по простому модулю, у которого не больше 2 решений, а значит на любой прямой лежит не больше 2 точек.

Теперь перейдём к полной задаче. Вначале опишем решение для $N \geqslant 6$. Выберем наибольшее простое P, которое строго меньше N. По теореме Бертрана-Чебышёва $P > \frac{N+1}{2}$. Расставим первых $P-1$ растаманов по клеткам $(i; \frac{-1}{i} \pmod P)$, P-го в клетку $(P-1, P+1)$, и оставшихся $N-P$ в клетки $(i; \frac{-1}{i-P} \pmod P)$.

Покажем, что никакие три точки не лежат на одной прямой. На каждой вертикали и горизонтали лежит не больше 2 точек. Рассмотрим все остальные прямые. Мы уже показали выше, что среди первых $P-1$ точек никакие три не лежат на одной прямой. Очевидно, что среди последних $N-P$ точек никакие три также не лежат на одной прямой, так как это просто копия первых $P-1$, сдвинутая на P вправо. Также понятно, что если взять одну точку среди первых $P-1$, P-ю точку

и одну точку среди последних $N - P$, то они не лежат на одной прямой, так как P-я точка правее одной, правее другой и выше обоих. Покажем для всех остальных троек, что они не лежат на одной прямой от обратного. Пусть это прямая $y = k \cdot x + b$. Нужно рассмотреть всего 4 случая:

- Две точки среди первых $P - 1$, одна точка среди последних $N - P$. Заметим, что у всех этих точек разные x по модулю P, иначе если совпали бы две, то у них одинаковый y, а на горизонтальной прямой лежит не больше двух точек. А это означает, что $k \cdot x^2 + b \cdot x \equiv -1 \pmod{P}$ имеет три различных решения, что не может быть правдой.
- Одна точка среди первых $P - 1$, две точки среди последних $N - P$. Симметричный предыдущему случай, доказывается аналогично.
- Две точки среди первых $P - 1$ и P-я точка. Можно использовать аналогичное доказательство, вместо этого рассмотрев прямые вида $x = k \cdot y + b$ и уравнение $k \cdot y^2 + b \cdot y \equiv -1 \pmod{P}$.
- P-я точка и две точки среди последних $N - P$. Если воспользоваться аналогичным доказательством, то находится ровно одно решение — это точки $(P - 1; P + 1)$, $(P + 1; P - 1)$ и $(2 \cdot P - 1, 1)$, однако поскольку $P > \frac{N+1}{2}$, то $2 \cdot P - 1 > N$, поэтому последняя точка никогда не будет поставлена.

Решения для $N < 6$ можно построить вручную.