Задача A: Плохой тетрис

Нужно составить самую высокую башню из множества полимино. При этом считается, что башня не падает под действием гравитации.

Подзадача 1: Квадраты

Можно просто ставить квадраты друг на друга любым способом. Ответом будет сумма длин сторон всех квадратов. Необходимо научиться считать длину стороны квадрата. Для этого можно найти строчку, в которой есть хотя бы одна клетка квадрата, и найти в этой строчке индекс первой такой клетки и последней такой клетки. Разность индексов, увеличенная на 1, будет равна ширине.

Подзадача 2: Прямоугольники

Стоит посчитать ширину и высоту прямоугольника. Ширина считается способом из первой подзадачи, высота равна количеству непустых строк. Далее, поскольку требуется построить максимально высокую башню, — нужно взять максимум из высоты и ширины и прибавить к ответу.

Подзадача 3: Полимино

В случае с произвольным полимино посчитать высоту и ширину приведённым выше способом не удастся. Однако, есть альтернативный способ. Поскольку известно, что на каждом поле есть ровно одна фигура полимино, достаточно найти самую верхнюю, самую нижнюю, самую правую и самую левую клетку, принадлежащую полимино. Исходя из их координат можно посчитать ширину и высоту полимино. Далее, аналогично подзадаче 2, необходимо взять максимум из высоты и ширины.

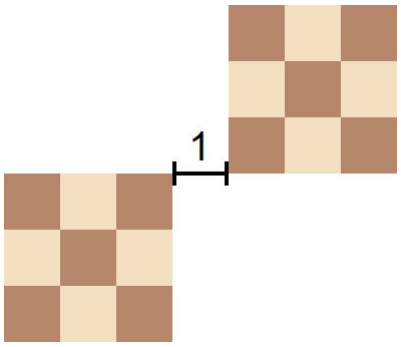
Задача B: Он вам не конь 3

На клетчатой плоскости разместили N шахматных досок разного размера, не касающихся ни сторонами, ни углами. В одну из клеток поставили шахматного коня. Определить, сколько досок он может посетить

Подзадача 1: N ≤ 2, M ≤ 100

Для начала разберем крайний случай, общий для всех подзадач: если конь начал в центральной клетке доски 3 на 3, то он никуда не может сходить. В этом случае ответ 1.

Иначе, конь может посетить все клетки стартовой доски и все клетки любой доски, куда он может попасть (кроме центральной клетки доски 3 на 3). Следовательно, нужно проверить, существует ли ход из клетки первой доски в клетку второй доски. Это можно сделать разными способами, например, перебрав все клетки первой доски и попробовав сходить из каждой. Или можно посчитать расстояние между границами досок, такой ход существует, если расстояние равно 1, то есть, расстояние между ближайшими двумя точками из разных досок равно 1.



Подзадача 2: N ≤ 100, M ≤ 100

Чтобы решить эту подзадачу, можно построить граф на клетках всего поля, лежащих на одной из досок (их не более 40000), соединить ребрами пары клеток, между которыми есть ход конем, и запустить поиск в глубину из стартовой клетки. После чего посчитать количество досок, которые достиг поиск в глубину.

Подзадача 3: N ≤ 1000, M ≤ 10^9

Для этой подзадачи необходимо для пары досок уметь считать, есть ли ход из одной в другую. Это можно сделать, вычислив расстояние между прямоугольниками. Чтобы решить подзадачу, построим граф на досках, соединив ребрами пары досок, между которыми есть ход. После чего запустим поиск в глубину из начальной доски и посчитаем количество посещенных вершин в графе.

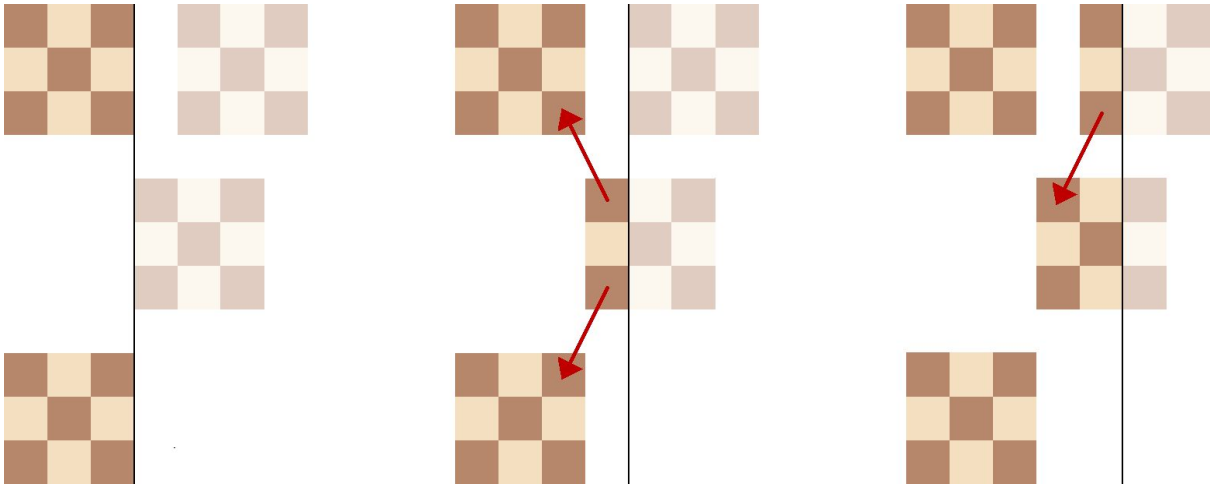
Асимптотика этого решения — $O(N^2)$, так как для построения ребер в графе нужно перебрать все пары вершин.

Подзадача 4: N ≤ 50000, M ≤ 100000

Покажем, что граф, построенный в предыдущей подзадаче, имеет не более $4N$ ребер. Разделим ребра на 2 группы: вертикальные и горизонтальные; вертикальные ребра будут соответствовать ходу конем на 2 клетки по вертикали и 1 по горизонтали,

горизонтальные ребра — ходу на 2 клетки по горизонтали и 1 по вертикали. Покажем, что вертикальных ребер не более $2N$ для горизонтальных все аналогично.

Начнем с поля, на котором нет ни одной клетки доски, и будем добавлять столбцы слева направо. Чтобы добавить столбец, добавим все клетки досок, лежащие на этом столбце. На каждом шаге у нас какие-то доски будут присутствовать полностью, какие-то — частично (в этом случае они все еще будут прямоугольниками, пусть и неправильного размера), остальных досок не будет вовсе. Будем также на ходу перестраивать граф, но чуть по другому: разрешим ход конем на 2 клетки по вертикали и 0 по горизонтали; это не поменяет конечный граф, но поможет строить его на ходу. Количество вертикальных ребер может увеличиться только в том случае, если после добавления столбца появилась новая доска, которой раньше не было. Причем каждая новая доска добавляет не более 2 ребер. Таким образом, всего вертикальных ребер будет не более $2N$.



Теперь нужно построить граф быстрее, чем за $O(N^2)$. У каждой доски возьмем только 4 угловые клетки, затем для каждого столбца и для каждой строки составим список, где будут лежать угловые клетки, лежащие в этом столбце или строке, по порядку. Чтобы найти все вертикальные ребра, исходящие из какой-то одной доски, посмотрим на строку на 2 выше самой верхней строки этой доски. Бинарным поиском найдем самую левую доску, имеющую угловую клетку в этой строке, в которую можно сходить, и самую правую такую доску. Во все доски между ними тоже можно сделать ход. Таким образом мы найдем все доски, в которые можно попасть из текущей шагом на 2 клетки вверх и 1 клетку вбок. Аналогично разберем три других хода: 2 клетки вниз и 1 вбок, 2 клетки влево и 1 вверх или вниз, 2 клетки вправо и 1 вверх или вниз.

Явно построив граф, запустим на нем поиск в глубину. Сложности такого решения — $O(N \cdot log(N))$, так как для каждой доски мы выполняем бинарный поиск. На все остальное уходит всего $O(N)$ действий.

Подзадача 5: N ≤ 50000, M ≤ 10^9

Для этой подзадачи реализуем сжатие координат. Всего имеется не более 2N строк и не более 2N столбцов, в которых есть угловая клетка. Значит, можно хранить не более

4N списков и делать бинарный поиск только в том случае, если нужная строка или столбец присутствует. Сложность остается такой же — $O(N \cdot log(N))$.

Задача C: Правильные печеньки

В окружность вписаны N правильных многоугольников. Требуется определить, сколько из них хотя бы частично не покрыты другими. Число вершин многоугольника будем называть его степенью.

Подзадача 1: N ≤ 2

Для N = 1 ответ 1. Иначе из двух многоугольников один покрывает другой, только если его степень кратна степени другого. В этом случает ответ 1, иначе 2.

Подзадача 2: N ≤ 1000

Одинаковые многоугольники друг друга покрывают, поэтому из них видно не более одного (если он не покрыт многоугольником другой формы). Будем рассматривать только различные многоугольники из набора. Пользуясь свойством из решения подзадачи 1, получаем, что ответ — это число многоугольников, чьи степени не делят степень никакого другого многоугольника из набора. В данной подзадаче можно для каждого многоугольника проверить это свойство, проверив делимость его степени на степени каждого из остальных. Сложность решения $O(N^2)$.

Подзадача 3: N ≤ 50000

Известно, что степени многоугольников не превосходят 100000. Заведем массив флагов, в котором флаг по индексу k будет означать, что в наборе нет многоугольника, который бы перекрыл k-угольник. Инициализируем флаги значением ИСТИНА.

Делители степени a_i произвольного многоугольника можно перебрать как числа вида i и a_i/i для всех i от 2 до $\lfloor\sqrt{a_i}\rfloor$ (где $[x]$ — целая часть x). Таким перебором для каждого многоугольника из набора отметим в массиве флагов значением ЛОЖЬ все степени многоугольников, которые в данном наборе можно перекрыть. После этого ответ находится подсчетом всех уникальных многоугольников, для степени которых флаг установлен в значение ИСТИНА. Сложность такого решения составляет $O(N \cdot \sqrt{M})$, где M — максимальная возможная по условию задачи степень многоугольника.

Альтернативное решение: Для каждого числа от 3 до M заранее запишем, есть ли многоугольник с таким количеством вершин. После чего для каждого многоугольника определим, делит ли его количество вершин количество вершин другого многоугольника. Если у текущего многоугольника a_i вершин, то многоугольник, покрывающий его, имеет $k \cdot a_i$ вершин, где k — натуральное число, большее 1. Переберем все k от 2 до $M \ div \ a_i$ и для каждого посмотрим в заранее составленном массиве, есть ли многоугольник с $k \cdot a_i$ вершинами. При этом важно рассматривать только многоугольники с различным числом вершин; если есть несколько одинаковых многоугольников, из них нужно рассмотреть только один.

Посчитаем время работы этого решения. Если все a_i различны, время работы составляет $O(M/a_1 + M/a_2 + \ldots + M/a_n)$. Это принимает наибольшее значение при $a_1 = 3,\ a_2 = 4,\ a_3 = 5,\ \ldots,\ a_n = n + 2$, тогда сложность работы решения составит $O(M/3 + M/4 + M/5 + \ldots + M/(N + 2))$. Можно доказать, что эта сумма имеет порядок $O(M \cdot log(N))$.

Задача D: Киберспортивный турнир

Игроки входят в игру и выходят из игры, а также обмениваются сообщениями: публичными и приватными. Каждое сообщение влияет на удачу игроков, необходимо уметь сравнивать значения удачи игроков.

Подзадача 1

В этой подзадаче нужно аккуратно реализовать все, что просят, кроме команды /quit, не опечататься в сообщениях и не допустить ошибок в реализации команд. Так как $A = B$, то можно считать, что мы не умножаем и делим удачу, а прибавляем и вычитаем ее; при этом нужно не забыть, что при $A = B = 1$ удачи всегда будут равны. Для поиска игроков

Подзадача 2

В этой подзадаче нужно добавить команду /quit и написать аккуратное сравнение рациональных чисел. Те, кто пишет на питоне, могут воспользоваться модулем fractions. В любом случае, можно хранить их в обычном типе вещественных чисел, так как ограничения небольшие.

Подзадача 3

Эта подзадача во многом сложнее предыдущей, кроме того, что здесь $A = B$. Это упрощает подсчет удачи: можно считать, что “gl” увеличивает удачу на 1, а “hf” уменьшает удачу на 1. Опять же, если $A = B = 1$, удачи всегда будут равны.

Подзадача 4

В этой подзадаче числа могут быть слишком большими, чтобы хранить их в обычном типе чисел с плавающей запятой. Чтобы считать удачу, воспользуемся следующим трюком: будем хранить не само значение удачи, а его логарифм. Логарифм — возрастающая функция, поэтому он сохраняет сравнение. Более того, если значение удачи равно A^x/B^y, то его логарифм равен $log(A) \cdot x - log(B) \cdot y$ (независимо от основания логарифма). Таким образом, можно не умножать на A и делить на B, а прибавлять $log(A)$ и вычитать $log(B)$. Сравнивать числа нужно с эпсилоном, меньшим $log(1.0001)$, то есть, нужно зафиксировать константу $\varepsilon < log(1.0001)$ и считать два числа равными, если они отличаются не более, чем на ε.

Удобно хранить числа не в виде одного вещественного числа, а в виде показателей степеней при A и B. То есть, если число равно A^x/B^y, можно явно хранить x и y. Это позволит написать решение без эпсилона: числа в таком виде можно проверять на равенство точно, а если они не равны, сравнить значения $log(A) \cdot x - log(B) \cdot y$.

Подзадача 5

Основная сложность этой подзадачи — эффективная реализация команды /say. Так как в игре могут быть $O(N)$ игроков и может быть отправлено $O(N)$ сообщений /say, выполнение всех команд вручную займет $O(N^2)$ времени. Поймем, как делать это быстрее.

Для начала заметим, что, так как нам нужно только сравнивать числа, то можно уменьшать удачу всех игроков на одно и то же число, и это не на что не повлияет. Значит, при выполнении команды /say увеличить удачу всех игроков в игре, кроме себя — это все равно, что уменьшить удачу себя и всех игроков не в игре.

Следующий шаг — воспользоваться тем, что игроки, которые сейчас не в игре, ни с кем не сравниваются. Это значит, что необязательно поддерживать актуальную удачу игроков, которые не в игре, а можно пересчитать ее в тот момент, когда игрок входит в игру.

Сделать это можно следующим образом: будем сохранять все команды /say в хронологическом порядке. Для каждого игрока, который вышел из игры, запомним, в какой момент времени он вышел. Когда игрок возвращается обратно в игру, нужно уменьшить его удачу на сумму всех изменений команд /say, произошедших, пока он был вне игры. Это можно сделать с помощью массива префиксных сумм, насчитывая его на ходу; после очередной команды /say добавим к массиву префиксных сумм его последний элемент (изначально 0), увеличенный на изменение удачи текущим сообщением.

Подведем итог: команды /write и /compare выполняются точно так же, обращаясь к удаче игрока напрямую. Команда /say вместо того, чтобы увеличить удачу всех игроков в игре, кроме автора, уменьшит удачу автора напрямую, а также будет внесена в массив префиксных сумм. При выполнении команды /quit игрок будет

запоминать текущую длину массива префиксных сумм, при повторном выполнении команды /join игрок вычтет из своей удачи сумму всех новых изменений, добавленных с тех пор, как он в последний раз вышел. При первом выполнении команды /join нужно вычесть из начальной удачи сумму всех изменений, то есть последний элемент массива префиксных сумм.

Решение работает за $O(N \cdot log(N))$ или за $O(N)$ в зависимости от того, какой структурой данных ищутся игроки по никам.

Задача E: Городки и роботы

На плоскости нарисован прямоугольник и отмечено N пар точек. Необходимо соединить парные точки кривыми так, чтобы кривые не пересекались между собой и не пересекали прямоугольник кроме концевых точек.

Подзадача 1: N = 1

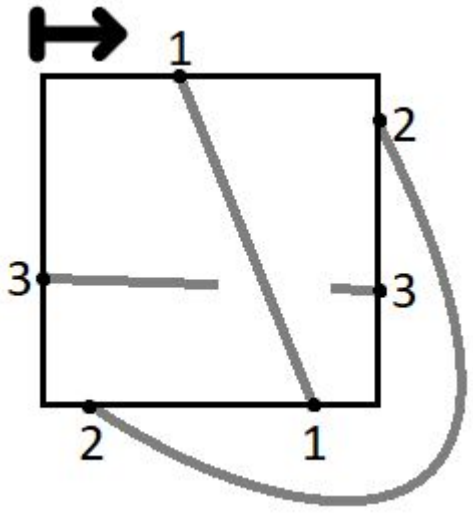
Есть всего одна пара точек. Если одна точка лежит строго внутри прямоугольника, а другая — строго снаружи него, то ответ NO, иначе — ответ YES.

Подзадача 2: N ≤ 2

Выполнить ту же проверку для каждой пары. Если для одной из пар одна точка лежит строго внутри прямоугольника, а другая — строго снаружи него, то ответ NO, иначе — ответ YES. Две кривые всегда можно нарисовать так, чтобы они не пересекались.

Подзадача 3: N ≤ 3

Кроме вышеуказанной проверки, нужно выполнить еще одну. Если есть 3 пары точек и каждая точка лежит на прямоугольнике, пойдем по кругу вдоль прямоугольника и запишем, какой паре принадлежит каждая точка. Если записано 123123 или аналогичное, то есть, каждая точка лежит между двумя точками из других пар, то нельзя соединить все пары кривыми без пересечений (задача «Три домика, три колодца»).



.

Подзадача 4: N ≤ 500

Обобщим предыдущие решения. Если есть пара точек такая, что одна лежит строго внутри прямоугольника, а другая — строго вне, то ответ NO.

Далее, заметим, что пары точек, в которых одна из точек лежит строго внутри или строго снаружи прямоугольника, на ответ не влияют. Можно провести кривые внутри этих пар самыми первыми, тогда каждая кривая не будет влиять на то, какие точки мы можем соединить.

Остались пары точек такие, в которых обе точки лежат на прямоугольнике. Если мы можем соединить точки внутри каждой из пар без пересечений, то каждая кривая будет лежать либо полностью внутри прямоугольника, либо полностью снаружи. Причем если две пары такие, что кривые внутри них могут пересечься, то их нужно пустить по разные стороны: одну кривую — внутри, другую снаружи.

Пойдём вдоль прямоугольника по кругу и для каждой встреченной точки запишем, в какой момент времени мы ее встретили. Тогда каждой интересующей нас паре точек будет сопоставлено два числа: время встречи первой и второй. Назовем их L_i и R_i, $L_i < R_i$. Тогда если найдутся две пары (L_i, R_i) и (L_j, R_j) такие, что $L_i < L_j < R_i < R_j$, то эти пары нужно пустить по разные стороны.

Теперь построим граф, вершинами которого будут пары точек. Между вершинами номер i и номер j проведем ребро в том и только в том случае, когда $L_i < L_j < R_i < R_j$ или $L_j < L_i < R_j < R_i$.

Осталось проверить граф на двудольность. Если граф двудолен, то ответ — YES; одна доля будет содержать все такие пары точек, которые мы соединим кривыми через внутреннюю часть прямоугольника, а другая доля — пары точек, которые мы соединим через наружную часть. Если граф не двудолен, ответ — NO.

Асимптотика такого решения — $O(N^2)$ из-за того, что количество ребер в таком графе может быть равно N^2.

Подзадача 5: N ≤ 100000

Сформулируем задачу по-другому. Будем идти вдоль прямоугольника по кругу. Для каждой пары точек скажем, что первая точка «открывает» ее, вторая — «закрывает» (первая и вторая в порядке обхода). Теперь наша задача — разбить все пары на две правильные скобочные последовательности.

Сделаем это следующим образом: будем поддерживать стек из «открытых» пар. Когда мы встретим первую точку из пары, добавим ее в стек. Когда мы встретим вторую точку из пары, если первая все еще в стеке, будем удалять точки с вершины стека, пока не найдем первую точку из этой же пары.

После завершения этого процесса мы будем знать, что все точки, которые мы не удалили из стека, можно соединить во внутренней части прямоугольника без пересечений. Осталось проверить, что все удаленные точки можно соединить во внешней части без пересечений; сделать это можно еще одним проходом со стеком.

Асимптотика такого решения — $O(N \cdot log(N))$ из-за сортировки, необходимой, чтобы рассмотреть точки в порядке обхода.

Альтернативное решение: Вернемся к решению задачи через графы. Во-первых, заметим, что если мы получим разбиение графа на две доли, мы можем проверить это разбиение на корректность за $O(N \cdot log(N))$. Во-вторых, заметим, что если заменить граф остовным лесом, у него будет ровно одно корректное разбиение на две доли с точностью до инвертирования компонент связности.

Будем строить остовный лес с помощью структуры данных “система непересекающихся множеств”. Точно так же, как и в первом решении, будем идти по точкам в порядка обхода прямоугольника, но теперь будем поддерживать упорядоченное множество открытых точек вместо стека. Каждый раз, когда точка «закрывает» свою пару, найдем все пары из других компонент, пересекающиеся с текущей. Для этого нужно выполнить запрос “На отрезке множества найти точку из компоненты, отличной от данной”, после чего слить компоненты текущей и найденной точек и повторять запрос до тех пор, пока он что-то находит.

Чтобы выполнять такой запрос, будем поддерживать второе упорядоченное множество — множество отрезков точек из первого множества, лежащих в одной компоненте. Это делается аккуратными операциями с множеством; чтобы поменять компоненту точки, удалим ее и добавим заново с другой компонентой.

Поскольку вы меняем компоненты сразу нескольких точек при слиянии компонент, удобно это делать системой непересекающихся множеств с одной эвристикой без сжатия путей. Таким образом, СНМ выполняет $O(N \cdot log(N))$ действий, каждое из которых влечет операции над упорядоченным множеством, поэтому решение работает за $O(N \cdot log^2(N))$.

Задача F. Web-котики

В этой задаче будет показан самый простой путь решения по мнению автора без рассмотрения всевозможных случаев, и ещё одно альтернативное решение.

Для начала поймем что $W = (A + k) \cdot N + B \cdot (N + 1)$, где k — это целочисленное отклонение ширины картинки от того что хочет заказчик, а B — ширина отступа.

Выразим ширину из этой формулы: $B = (W - (A + k) \cdot N) / (N + 1)$. Так как в задаче все ширины (и картинок, и отступов) целочисленны, то рассмотрим, в каком случае B является целым числом. Немного преобразуем нашу формулу:
$B = (W - (A + k) \cdot N) / (N + 1) = (W + (A + k) - (A + k) \cdot (N + 1))) / (N + 1)$. Отсюда мы видим, что B целочисленно тогда и только тогда, когда $W + (A + k)$ делится на $N + 1$ без остатка.

Определим множество возможных значений для $A + k$ таких, что при них значение B — целое.

Таким множеством будет
$A + k \, \varepsilon \, \{N + 1 - W \, mod \, (N + 1) + (N + 1) \cdot t \mid t - \text{целое число}\}$. ***(1)***

В этом множестве присутствуют значения, применяя которые получаются некорректные значения в рамках нашей задачи, такие как нулевые или отрицательные значения ширины картинок/отступов.

Заметим что минимальное возможное валидное значение $A + k$ достигается при $t = 0$.

Найдем ближайшие к A значения $A + k$: Из формулы
$A + k = N + 1 - W \, mod \, (N + 1) + (N + 1) \cdot t$ выведем t:
$t = ((A + k) - (N + 1 - W \, mod \, (N + 1)) / (N + 1)$.

Вычислим это значение подставив вместо $k = 0$ и округлим по стандартным математическим правилам. Мы получили такое целое значение t, при котором, при подстановке в формулу (1) получается ближайшее к значению заказчика значение ширины картинок, при котором ширина отступов тоже целая.

Теперь нам необходимо найти такое максимальное число $A + k$ между значениями $t = 0$ и найденном t, которое не порождает невалидных значений ширины картинок и отступов, что и является ответом. С этой задачей прекрасно справляется бинарный поиск.

Если же ни при одном значении $A + k$ из данного промежутка для t, значения ширины картинок и отступов не становятся валидными, то ответ “NO”.

Альтернативное решение: Выразим ширину из формулы:
$B = (W - (A + k) \cdot N) / (N + 1)$. Немного преобразуем эту формулу:

$B = (W - (A + k) \cdot N) / (N + 1)$

$B = (W + (A + k) - (A + k) \cdot (N + 1))) / (N + 1)$

$B = (W + A + k) / (N + 1) - A - k$

Отсюда мы видим, что B целочисленно тогда и только тогда, когда $W + A + k$ делится на $N + 1$ без остатка.

Не все k дают корректные решения; надо, чтобы ширина картинок и ширина отступов была положительна. Это дает нам условия $A + k \geq 1$ и $B \geq 1$. Преобразуя второе неравенство, получим

$(W + A + k) / (N + 1) - A - k \geq 1$

$W + A + k - (A + k) \cdot (N + 1) \geq N + 1$

$W - AN \geq N + 1 + kN$

$k \leq (W - AN - N - 1)/N$

Так как k целое, можно даже сказать $k \leq [(W - AN - N - 1)/N]$ (где $[x]$ — целая часть x).

Таким образом, нам нужно найти число k такое, что $l \leq k \leq r$, где $l = 1 - A$ и $r = [(W - AN - N - 1)/N]$, и $W + A + k$ делится на $N + 1$ без остатка, то есть $k \equiv - A - W \ (mod\ (N + 1))$. Из всех таких k нам нужно самое близкое к нулю, если таких несколько — положительное.

Очевидными кандидатами являются $(- A - W)\ mod\ (N + 1)$ и $((- A - W)\ mod\ (N + 1)) - N - 1$, самые близкие к нулю числа с нужным остатком (будем считать, что $x\ mod\ y$ всегда лежит на отрезке от 0 до $y - 1$). Эти два числа могут не входить в отрезок от l до r, в этом случае ответ будет рядом с концом отрезка. Ближайшее к левому концу отрезка число — или $l - ((l + A + W)\ mod\ (N + 1))$, или $l - ((l + A + W)\ mod\ (N + 1)) + N + 1$. Ближайшее к правому концу отрезка число — $r - ((r + A + W)\ mod\ (N + 1))$.

Итак, у нас есть 5 кандидатов:

- $k = (- A - W)\ mod\ (N + 1)$
- $k = ((- A - W)\ mod\ (N + 1)) - N - 1$
- $k = l - ((l + A + W)\ mod\ (N + 1))$
- $k = l - ((l + A + W)\ mod\ (N + 1)) + N + 1$
- $k = r - ((r + A + W)\ mod\ (N + 1))$

Нужно взять значения k, лежащие на отрезка от l до r, из них — ближнее к нулю, если таких несколько — положительное. После чего вывести $A + k$. Если подходящего значения среди кандидатов вообще нет, вывести “NO”.

Задача G: Любимые бутерброды

Для одного бутерброда нужно A хлеба и B колбасы. Когда Вале не хватает ингредиентов, он идет в магазин и покупает наименьшее количество хлеба и колбасы, чтобы ему хватило на один бутерброд, но хлеб продается батонами веса X, а колбаса — палками веса Y. Сколько раз Валя ходил в магазин, если он съел N бутербродов?

Подзадача 1: A, B, X, Y, N ≤ 100

Для начала, если $A \geq X$ или $B \geq Y$, то после каждого бутерброда нам нужно будет идти в магазин, и ответ будет равен N. Все последующие решения предполагают, что $A < X$ и $B < Y$.

Реализуем то, что от нас просят в условии. Запустим цикл из N итераций и будем поддерживать текущее количество хлеба и колбасы. Если хлеба меньше A или колбасы меньше B, будем покупать хлеб, пока его не станет хотя бы A, и колбасу, пока ее не станет хотя бы B. За каждую такую покупку увеличим ответ на 1.

Подзадача 2: A, B, X, Y ≤ 100, N ≤ 10^9

Пусть мы съели $X \cdot Y$ бутербродов. Тогда мы съели $A \cdot X \cdot Y$ хлеба и $B \cdot X \cdot Y$ колбасы. Так как мы покупаем хлеб в количествах, кратных X, а колбасу — в количествах, кратных Y, то после съедения $X \cdot Y$ бутербродов у нас останется ровно 0 хлеба и колбасы. Если N больше, чем $X \cdot Y$, посчитаем циклом, сколько раз мы сходили в магазин, чтобы съесть $X \cdot Y$ колбасы, после чего мы сможем есть бутерброды пачками по $X \cdot Y$ штук, для каждой увеличивая ответ на одно и то же число. После чего останется $N \bmod (X \cdot Y)$ несъеденных бутербродов, которые мы сможем съесть также циклом. Сложность такого решения — $O(X \cdot Y)$.

Альтернативное решение: будем поддерживать состояние — количество хлеба и колбасы в запасе. Изначально у нас 0 хлеба и 0 колбасы, перед походом в магазин или хлеба меньше A, или колбасы меньше B, а после похода в магазин — хлеба не более $A + X$, колбасы не более $B + Y$. Таким образом, возможных состояний не более $(A + X) \cdot (B + Y)$. Будем есть бутерброды простым циклом, запоминая, сколько походов в магазин было сделано до каждого из состояний, пока не вернемся в состояние, где мы уже были. Зная длину цикла и количество походов в магазин за этот цикл, мы сможем есть бутерброды пачками, пока не останется съесть меньше, чем длина цикла, что в свою очередь меньше $(A + X) \cdot (B + Y)$. Сложность такого решения — $O((A + X) \cdot (B + Y))$.

Подзадача 3: A, B ≤ 100, X, Y, N ≤ 10^9

Усовершенствуем решение для предыдущей подзадачи. А именно, будем поддерживать количество хлеба k и количество колбасы t. Чтобы ускорить предыдущее решение, будем “перепрыгивать” между состояниями, съедая несколько бутербродов за раз и считая при этом число походов в магазин.

Первая оптимизация, которую мы применим, заключается в следующем: если у нас хватает ингредиентов на несколько бутербродов, съедим их все вместо того, чтобы есть по одному. Из этого получится, что на каждом шаге у нас либо хлеба меньше, чем A, либо колбасы меньше, чем B. Это уменьшает число состояний: на каждом шаге либо $t < A$ и $k < B + Y$, либо $t < A + X$ и $k < B$, и всего возможных состояний меньше, чем $A \cdot (B + Y) + (A + X) \cdot B$.

Этого недостаточно, может возникнуть случай, когда одного из ингредиентов слишком много. Для этого нужна вторая оптимизация. На каждом шаге посмотрим на ингредиент, которого больше, и посчитаем, сколько бутербродов нужно съесть до того, как придется покупать этот ингредиент. До покупки этого ингредиента нам придется ходить в магазин только за другим ингредиентом, и мы можем быстро посчитать, сколько походов в магазин это займет.

Например, если у нас имеется t хлеба и k колбасы, то до следующей покупки хлеба мы съедим $t\ div\ A$ бутербродов. За это время мы съедим $B \cdot (t\ div\ A)$ колбасы и сходим в магазин m раз, где m — наименьшее целое число, подходящее под условие $k - B \cdot (t\ div\ A) + m \cdot Y \geq 0$, что равносильно $m \geq (B \cdot (t\ div\ A) - k) / Y$. Отсюда можно найти число походов в магазин m.

Это уже дает нам решение, которое с одной стороны работает не более $A \cdot (B + Y) + (A + X) \cdot B \leq 2AB + 2(A + B) \cdot max(X, Y)$ шагов, и с другой стороны ест $max(X\ div\ A, Y\ div\ B) \geq max(X\ div\ max(A, B), Y\ div\ max(A, B)) = max(X, Y)\ div\ max(A, B)$ бутербродов за раз, что закончит выполняться за $O(N \cdot max(A, B) / max(X, Y))$ шагов. Таким образом, решение отработает за $O(min(2AB + 2(A + B) \cdot max(X, Y),\ N \cdot max(A, B) / max(X, Y)))$ шагов. Максимум достигается при наибольших A, B и N и таком $max(X, Y)$, что левая часть равна правой. Это дает асимптотику решения примерно $O(\sqrt{N} \cdot max(A, B))$.