Задача A. Снова ИИ

В первой подзадаче (**34 балла**) достаточно перебрать все числа от 1 до N и для каждого числа X проверить, верно ли, что $N \ div \ (N \ div \ X) = X$.

Пусть X супер-обратимо для N, и $Y = N \ div \ X$. Тогда выполняется $X = N \ div \ Y$. К тому же, Y супер-обратимо для N: если поделить N нацело на обе части равенства $X = N \ div \ Y$, получим $N \ div \ X = N \ div \ (N \ div \ Y)$, отсюда $Y = N \ div \ (N \ div \ Y)$.

Таким образом, все супер-обратимые числа образуют пары (X, Y) такие, что $N \ div \ X = Y$ и $N \ div \ Y = X$. Отсюда $X \cdot Y \leqslant N$. Значит, хотя бы одно число в паре не превосходит $\sqrt{N}$. Получаем решение для второй подзадачи (**34 балла**): перебираем все X от 1 до $\left\lfloor \sqrt{N} \right\rfloor$, проверяем каждое число X и каждое число $N \ div \ X$.

Чтобы решить третью подзадачу (**32 балла**), нужно более подробно посмотреть, какие числа образуют пары (X, Y). Оказывается, любое число $X \leqslant \left\lfloor \sqrt{N} \right\rfloor$ является супер-обратимым. Докажем это от противного.

1. Пусть $X \leqslant \left\lfloor \sqrt{N} \right\rfloor$ и $N \ div \ (N \ div \ X) < X$. Так как $N \ div \ X = Y$, выполняется $X \cdot Y \leqslant N$. Если $N \ div \ Y < X$, то $X \cdot Y > N$. Противоречие.
2. Пусть $X \leqslant \left\lfloor \sqrt{N} \right\rfloor$ и $N \ div \ (N \ div \ X) > X$. Так как $N \ div \ X = Y$, выполняется $X \cdot Y \leqslant N$ и $X \cdot (Y + 1) > N$. Если $N \ div \ Y > X$, то $Y \cdot (X + 1) \leqslant N$. Отсюда получаем, что $Y \cdot (X+1) < X \cdot (Y+1)$, следовательно, $Y < X$, или же $Y+1 \leqslant X$. Но тогда $X \cdot (Y+1) \leqslant X \cdot X \leqslant N$, это противоречит тому, что $X \cdot (Y + 1) > N$.

Таким образом, для $X \leqslant \left\lfloor \sqrt{N} \right\rfloor$ остаётся только один вариант — $N \ div \ (N \ div \ X) = X$. Получаем, что все числа $X \leqslant \left\lfloor \sqrt{N} \right\rfloor$ супер-обратимы для N.

Итак, существует $\left\lfloor \sqrt{N} \right\rfloor$ пар супер-обратимых чисел (X, Y). Ответ равняется $2 \cdot \left\lfloor \sqrt{N} \right\rfloor$, но из этого числа нужно вычесть единицу, если есть $N \ div \ \left\lfloor \sqrt{N} \right\rfloor = \left\lfloor \sqrt{N} \right\rfloor$, так как в этом случае есть пара из двух одинаковых элементов $(\left\lfloor \sqrt{N} \right\rfloor, \left\lfloor \sqrt{N} \right\rfloor)$.

Задача B. Как пройти в столовую

Для решения первой подзадачи (**25 баллов**) нужно сделать то, что просят — просто перемещаемся по лабиринту, аккуратно проверяя каждый шаг в лабиринте. Время работы составит $O(H \cdot W + L \cdot \max(H, W))$.

Решение второй подзадачи (**25 баллов**) также требует простого перемещения по лабиринту, однако теперь нужно дополнительно держать счётчик ключей, не забыть, что каждый ключ мы берём по одному разу, а каждую дверь открываем ровно один раз. Сложность решения аналогичная $O(H \cdot W + L \cdot \max(H, W))$.

В третьей подзадаче (**25 баллов**) обратите внимание, что поле не укладывается в статический массив. Нужно либо хранить массив строк динамической длины, либо записать поле в одну строку и аккуратно переводить координаты из двумерных в одномерные. Чтобы решить подзадачу, для каждой пустой клетки для каждого направления заранее посчитаем длину шага. Если считать в правильном направлении, это можно сделать за $O(1)$ для каждой клетки: ответ для клетки либо 0, если дальше по направлению идёт стена или граница поля; иначе — 1 плюс ответ для следующей клетки. Получаем время работы для подзадачи $O(H \cdot W + L)$.

Для решения четвёртой подзадачи (**25 баллов**) поймём, что перемещение по пустым клеткам бесполезно — нам важно лишь искать ближайшую из трёх важных клеток: стена, дверь, ключ (можно считать, что поле снаружи ограничено сплошной стеной). Таким образом, по строкам и по столбцам можно поддерживать только важные клетки, чтобы быстро искать следующую по направлению. Например, в языке C++ это можно делать при помощи структуры данных `set` и методов `lower_bound`, `upper_bound`. Для текущей точки в зависимости от направления ищем ближайшую стену/ключ/дверь; если ключ — забираем и удаляем; если дверь и можем открыть — открываем

и удаляем; иначе наш шаг закончился. Каждый ключ и каждую дверь мы удалим не более одного раза (при этом не забывайте, что удаление происходит из ряда и из столбца). Получаем решение за $O(H \cdot W \cdot (log(H) + log(W)) + L \cdot log(\max(H, W)))$.

Жюри умеет решать эту задачу за $O((H \cdot W + L) \cdot (\alpha(H) + \alpha(W)))$ с помощью *системы непересекающихся множеств*.

Задача C. Baba is You

В первых двух подзадачах (**20 и 18 баллов**) требуется реализовать то, что просят в условии. Чтобы ответить на запрос, будем переходить по правилам из запроса. Если для какого-то слова есть два правила, выводим «`TOO COMPLEX`». Если мы прошли более 1000 шагов и не попали в слово без правил, выводим «`INFINITE LOOP`».

В третьей подзадаче (**20 баллов**) нужно сделать то же самое, но хранить правила в виде графа в хэш-таблице или любой другой структуре данных. Сложность алгоритма должна быть $O(N \cdot Q \cdot DataStructure)$.

В четвёртой подзадаче (**22 балла**) нужно быстро ходить по графу, в котором из каждой вершины ведёт не более одного ребра. Для этого воспользуемся эвристикой *сжатия путей*. Чтобы ответить на запрос, пройдём по пути до листа, запоминая путь (листом здесь называется вершина, из которой не исходит ребро). После этого перенаправим все рёбра из вершин пути в этот лист. Добавление новых рёбер будем выполнять как обычно.

Это аналогично эвристике сжатия путей, использующейся в *системе непересекающихся множеств* (далее — СНМ), с тем только различием, что мы работает с ориентированными рёбрами. Этот алгоритм работает за $O(N + Q \cdot (logN + logQ))$, доказательство асимптотической сложности здесь не приводится, так как оно совпадает с доказательством для СНМ; его можно найти самостоятельно.

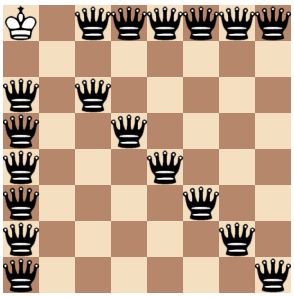
В пятой подзадаче (**20 баллов**) нужно дополнить решение четвёртой подзадачи, научившись обрабатывать ошибки. Чтобы обрабатывать циклы, можно поступать так же, как в СНМ: чтобы провести ребро из листа u в вершину v, запустим сжатие путей из v, и проведём ребро из u в вершину, куда ведёт путь из v. Если образовался цикл, то получится, что мы проведём петлю из u в u. Таким образом, сжатие пути нужно останавливать в листе или в вершине с петлёй, точно так же, как это делается в СНМ.

Чтобы обрабатывать ошибки «`TOO COMPLEX`», будем поддерживать множество *сломанных* вершин — так будем называть вершины, запрос к которым ведёт к этой ошибке. Сломанными являются вершины, из которых исходит два различных ребра, а также все вершины, из который они достижимы. Заметим, что если вершина стала сломанной, она никогда не перестанет быть таковой, так как из неё всегда будет существовать путь к вершине, из которой ведёт два различных ребра. Как только мы «ломаем» вершину, обойдём все еще не сломанные вершины, из которых одна достижимы, и пометим их как сломанные. Для этого будем хранить граф обратных рёбер. Каждую вершину мы пометим как сломанную не более одного раза, поэтому асимптотическая сложность алгоритма останется такой же.

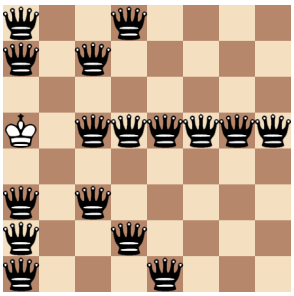
Задача D. Он вам не ферзь

Первую подзадачу (**14 баллов**) труднее решить формулой, чем написать решение второй подзадачи. Однако формула для первой подзадачи используется в последней подзадаче, поэтому приведём её здесь.

1. Если единственный король стоит в углу доски, то ответ равен $M \cdot 3 - 6$.



2. Если единственный король стоит на стороне доски, то ответ равен $M \cdot 3 - 8$.

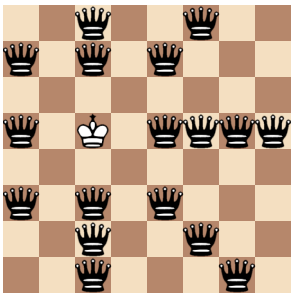


3. Если единственный король стоит в координатах (x, y) не в углу и не в стороне доски (считается, что $(1, 1)$ и (M, M) — координаты противоположных угловых клеток), то ответ равен

$$(M-3) \cdot 2 + min(x-2, y-2) + min(M-x-1, y-2) +$$
$$+min(x-2, M-y-1) + min(M-x-1, M-y-1).$$

Эту формулу можно упростить до

$$M \cdot 3 - 11 + min(x-1, y-1, M-x, M-y) \cdot 2.$$



Чтобы решить вторую подзадачу (**19 баллов**), нужно перебрать все клетки, для каждой клетки перебрать всех королей и проверить, что ферзь из этой клетки бьёт всех королей и сам не находится под боем. Ферзь с координатами (x, y) бьёт короля с координатами (x_i, y_i), если выполняется одно из четырёх условий: $x = x_i$, $y = y_i$, $x + y = x_i + y_i$ или $x - y = x_i - y_i$. Ферзь не находится под боем, если $|x - x_i| \geqslant 2$ или $|y - y_i| \geqslant 2$. Такое решение работает за $O(M^2 \cdot N)$.

Решить третью подзадачу (**20 баллов**) можно несколькими способами, самый простой из них — написать то же самое, что и во второй подзадаче, но обрывать проверку позиции ферзя как только мы находим короля, который не подходит под условия. Оказывается, что такое решение работает за $O(M^2 + M \cdot N)$, так как существует не более $4 \cdot M$ возможных позиций ферзя, которые бьют первого короля; во всех остальных случаях мы выйдем из цикла на первой же итерации.

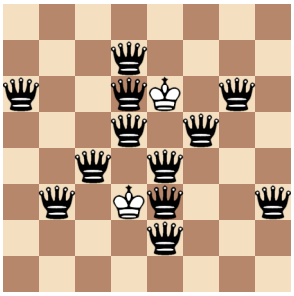
Чтобы решить четвёртую подзадачу (**28 баллов**), нужно перебирать только те позиции, которые бьют первого короля (опять же, их не более $4 \cdot M$), а также быстро проверять выполнение всех

условий. Для этого мы заведём 4 массива счётчиков: первый будет считать количества значений x_i, второй — количества значений y_i, третий — количества значений $x_i + y_i$, четвёртый — количества значений $x_i - y_i$. Также внесём координаты всех королей и всех смежных им клеток в хэш-таблицу.

Чтобы проверить, что ферзь с координатами (x, y) бьёт всех королей, нужно посчитать суммарное количество значений x, y, $x+y$ и $x-y$ в соответствующих массивах. Это число должно равняться n. Кроме того, клетка ферзя должны быть не занята королём и не быть смежной королю. Это можно проверить, сделав один запрос к хэш-таблице. Итого асимптотика решения $O((M+N) \cdot HashTable)$.

Решение пятой подзадачи (**19 баллов**) разбивается на случаи:

1. Есть ровно один король. Формула для этого случая приведена в начале разбора;
2. Есть два короля, не лежащих на одной горизонтали, вертикали и диагонали. Возможно, королей больше, но нам нужно найти двух таких. Существует не более 12 позиций ферзя, бьющих обоих этих королей. Переберём эти позиции и проверим их корректность для всех королей;



3. Все короли лежат на одной прямой (на одной горизонтали, вертикали или диагонали). Если поставить ферзя не на эту прямую, он должен бить первых двух королей. Аналогично предыдущему случаю, нужно проверить не более 12 позиций. Если ферзя поставить на эту прямую, то он автоматически будет бить всех королей; нужно только проверить, что он не стоит рядом с королём. Посчитаем количество клеток на этой прямой и вычтем количество клеток на этой прямой, занятых королём или смежных с королём. Это можно сделать, сложив все такие клетки в хэш-таблицу;
4. Любые два короля лежат на одной прямой, но нет прямой, на которой лежат все короли. Можно рассмотреть любую пару королей, перебрать не более 12 позиций ферзя, не лежащих на общей для них прямой; после чего добавить третьего короля и проверить не более 3 позиций, лежащих на этой прямой и бьющих третьего короля.

Решение 4-го случая можно использовать во 2-м случае, чтобы не искать пару королей, не лежащих на одной горизонтали, вертикали или диагонали. Или же в 4-м случае можно просто рассмотреть все такие расположения королей вручную: есть 8 таких конфигураций для трёх, четырёх или пяти королей.

Итоговая асимптотика решения $O(N \cdot HashTable)$.

Задача E. Сила есть, ум тоже нужен

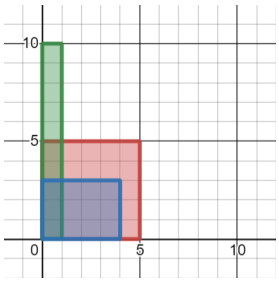
В первой подзадаче (**21 балл**) нужно заметить, что возможных проблем не более 100: проблемы — это пары (x, y), где x и y — натуральные числа от 1 до 10. Нужно перебрать все проблемы, и для каждого человека проверить, какие проблемы он или его подчинённые могут решить. Сделать это можно с помощью поиска в глубину, асимптотическая сложность такого решения $O(N^2 \cdot A \cdot B)$.

Во второй подзадаче (**26 баллов**) нужно заметить, что хоть количество проблем большое, количество значений x не превосходит 100. Значит, перебрать все значения x, и для каждого человека посчитать наибольшее значение y такое, что этот человек может решить проблему (x, y). Если $A_i < x$, то i-й человек не может сам решить проблему с физической сложностью x; если же $A_i \geqslant x$, то этот человек может сам решить проблему (x, B_i). С помощью поиска в глубину посчитаем максимум по всем подходящим значениям B_j, это и будет наибольший y для фиксированного x. Итого решение за $O(N^2 \cdot A)$.

В третьей подзадаче (**23 балла**) нужно отказаться от поиска в глубину. Переберём всех людей циклом от N-го до 1-го. Для каждого человека будем хранить массив из 100 значений — наибольший возможный y для каждого x (по умолчанию 0). Для i-го человека проинициализируем первые A_i значений числом B_i. Затем рассмотрим всех непосредственных подчинённых и обновим текущий массив с помощью массивов подчинённых, заменяя каждое число максимумом. Поскольку суммарное количество непосредственных подчинённых равно $N-1$, решение работает за $O(N \cdot A)$.

Четвёртую подзадачу (**14 баллов**) можно решить с помощью сжатия координат за $O(N^2)$, так как различных значений A не более N. Однако есть другое решение, которое более похоже на решение пятой подзадачи. Заметим, что множество проблем, которое может решить человек или любой из его подчинённых, представляет из себя объединение прямоугольников. Если ввести систему координат и представить i-го человека как прямоугольник с противоположными вершинами $(0,0)$ и (A_i, B_i), то ответ для i-го человека будет равен площади объединения прямоугольников на поддереве.

На рисунке проиллюстрирован пример из условия:



Давайте хранить форму фигуры, задающую ответ i-го человека. Фигура ограничена снизу осью OX и слева осью OY, поэтому достаточно хранить только точки, являющиеся верхними правыми вершинами прямоугольников. Иными словами, будем хранить *выпуклую оболочку* точек (A_i, B_i). Каждый человек инициализируется одной точкой (A_i, B_i). Чтобы посчитать множество точек для человека, нужно объединить его точку и все множества его детей. Это можно сделать по-разному, например, храня точки по возрастанию координаты A_i и строить текущее множество, добавляя туда точки одну за другой. Если в множестве точек лежит точка (A_i, B_i), и мы пытаемся добавить покрывающую её точку (A_j, B_j) с $A_j \geqslant A_i$ и $B_j \geqslant B_i$, удалим точку (A_i, B_i). Если мы пытаемся добавить покрытую точку (A_j, B_j) с $A_j \leqslant A_i$ и $B_j \leqslant B_i$, просто не будем её добавлять.

Можно реализовывать это решение по-разному, но поскольку суммарное количество подчинённых ограничено сверху числом $\frac{N(N-1)}{2}$, решение будет иметь как минимум квадратичную сложность. Можно добиться сложности $O(N^2 logN)$, используя упорядоченное множество, которое есть в стандартной библиотеке некоторых языков программирования.

Чтобы решить пятую подзадачу (**16 баллов**), воспользуемся приёмом «*сливаемые структуры*». Посчитаем размер каждого поддерева. Известно, что количество точек в множестве i-го человека не превосходит размера её поддерева. Чтобы получить ответ для i-го человека, нужно слить множество из одной точки (A_i, B_i) со всеми множествами непосредственных подчинённых i-го человека. Если у i-го человека нет подчинённых, это делается тривиально.

Давайте возьмём за основу множество непосредственного подчинённого с наибольшим размером поддерева. Сделаем его своим множеством, не копируя полностью (например, с помощью функции `move` на языке C++). Если для каждого человека хранится его множество точек, можно скопировать себе множество по указателю или по индексу. После чего переберём все точки во всех остальных множествах, в том числе точку (A_i, B_i). Каждую из них вставим в текущее множество. Здесь важна скорость вставки, вставку нужно выполнять за $O(logN)$. При вставке можно удалить какое угодно количество точек из множества, так как каждая точка удалится не более одного раза.

Получается, что мы за $O(1)$ перемещаем все точки из подчинённого с наибольшим размером поддерева, затем за $O(logN)$ перемещаем каждую оставшуюся точку по одной. Однако, каждый раз когда мы перемещаем точку за $O(logN)$, размер поддерева, где она находится, увеличивается хотя бы вдвое, так как точка перемещается из меньшего поддерева в сумму меньшего с большим. Итого у нас будет не более $O(logN)$ перемещений каждой точки, следовательно, итоговая асимптотика решения $O(Nlog^2 N)$.

Задача F. Трудный выбор

Для первой подзадачи (**10 баллов**) достаточно симулировать процесс и проверять, если для текущей группы по считалочке оказалось, что мы оказались не на стартовой позиции, то это ответ.

Для второй подзадачи (**30 баллов**) нужно свернуть симуляцию в следующее: для текущего b найти первый i такой, что $v \not\equiv 1 \pmod{a_i + 1}$. Запустим перебор, пока не найдём таковой.

Третья и четвёртая подзадачи (**16 баллов каждая**) требуют от нас как-то оптимизировать вычисления: либо в сторону быстрых ответов на запросов, жертвуя асимптотикой по N, либо наоборот - отвечать не быстро, но что-то быстро вычислить по a_i.

Что мы можем сделать в первом случае — давайте вычислим заранее для каждого возможного запроса такую функцию, как mex - первый не встречающийся индекс в наборе, рассмотрев для всех a_i кратные им числа в нашем диапазоне. Получится для каждого числа мы вычислим набор из индексов делителей, а т.к. нам нужен минимальный номер, где стоит не делитель, то mex набора будет ответом.

Если это делать «в лоб», то мы получим вердикт TLE, т.к. для малых a_i очень много кратных. Поэтому разделим числа на две группы — те, по которым mex мы вычисляем, и те, которые мы проверим перебором в запросе. Не трудно подсчитать, что оптимальная граница пройдёт рядом с числом 220, и в таком случае каждое из вычислений у нас займёт по не более чем $2,2 \cdot 10^7$ операций. Ответом будет является $\min(p_1, ..., p_k, mex(b_i - 1))$, где p_j — первая позиция, где встречается j-й не делитель $(b_i - 1)$, меньший 220.

Теперь рассмотрим случай, когда мы хотим отвечать быстро для большого набора чисел, но с малым числом запросов. Давайте сохраним для каждого числа из a_i только первую позицию, где он встретился, а если среди них его не было - число большее n. Ответ на запрос будем вычислять следующим образом - найдём для текущего числа $(b_i - 1)$ все его делители - это точки, которые мы выкалываем из набора, на котором нам нужно найти минимум. Так давайте подсчитаем минимум по отрезкам между этими числами и минимальный из них и будет ответом. Минимум на отрезке можно вычислить за log от длины, найти все делители - за корень от числа, что позволит нам на каждый из запросов отвечать не хуже чем за 2000 операций, а предподсчёт можно сделать не хуже чем за $4 \cdot 10^6$ операций (построение дерева отрезков).

Чтобы решить пятую подзадачу (**15 баллов**), нужно обратить внимание на следующий факт — максимальное число делителей для возможного запроса не превышает 240 (число 720720). Научимся решать за $Q \cdot 241$. Т.к. у нас не более 240 различных делителей, то, перебрав первые 241 различных чисел a_i мы гарантированно завершим перебор. Т.е. оставим в массиве a только различные, оставив только первое вхождение и записав его позицию. Теперь, если мы будем запускать перебор из второй группы он будет работать не за $O(NQ)$, а лишь за $N + Q \cdot 241$.

Для шестой подзадачи (**13 баллов**) давайте будем кешировать наши ответы, и тогда мы ещё сильнее улучшаем время работы до $O(N + Q + X)$, где X — сумма функции количества делителей среди различных b_i. Она в реальности будет ещё быстрее, потому как большинство чисел не будет иметь всех своих делителей на префиксе. Чтобы решение на Python проходило шестую подгруппу нужно оптимизировать I/O при помощи библиотеки *sys*.

Однако есть другое решение для шестой подгруппы — заметим, что если число b_i делится на префикс из чисел a_j, то оно кратно их НОК. При этом если число a_j не увеличивает величину НОК, то оно никогда не будет ответом. А значит мы можем убрать все такие a_j, а также теперь нам достаточно префикса размером $\log b_i$, чтобы дать ответ. В таком случае мы получаем асимптотику $O(N + Q \cdot \log b_i)$, что позволяет решать эту задачу для гораздо больших ограничений a_i и b_i.

Задача G. Ulearn

В первой подзадаче (**17 баллов**) $B < A$, поэтому во время выполнения 2-го этапа Паша не может начать работать над новой задачей; т.е. прежде чем брать следующую задачу, мы для текущей должны выполнить этапы 1-3. Получается, $2A + B$ — это величина времени, за которое делается ровно одна задача, после чего на первый этап выходит следующая, и так далее. Нужно сделать N задач, каждая занимает $2A + B$ времени, и в конце ещё B времени уходит на 4-й этап последней задачи. Итого ответ — $N(2A + B) + B$.

Для решения второй подзадачи (**15 баллов**) нужно отдельно рассмотреть случай, когда $B = A$.

В таком случае мы можем чередовать для пары задач нечётные и чётные этапы (для первой задачи мы делаем 2,3,4 этапы, в то время как у второй выполняются 1,2,3). Если N чётное, то у нас есть $\frac{N}{2}$ пар задач, на каждую пару тратится $5A$ единиц времени. Когда вторая задача выполняет 4-й этап, мы можем для новой пары выполнять 1-й этап первой задачи, а значит мы получаем ответ равный $\frac{N}{2} \cdot 4A + A = (2N + 1)A$. Если N нечётное, то нужно в конце добавить еще $3A$ времени на последнюю задачу. Получается ответ равный $\frac{N+1}{2} \cdot 4A = (2N + 2)A$. Нетрудно заметить, что и чётный и нечётный случай объединяются в одну формулу $(2N + 1 + (N \ mod \ 2))A$.

Чтобы решить третью подзадачу (**16 баллов**), рассмотрим отдельно 2 случая: $B = 2A$ и $A < B < 2A$.

1. Пусть $B = 2A$. Теперь мы можем параллельно работать уже с тремя задачами. Время выполнения одной такой группы составит $8A$ единиц времени, причём следующая группа задач может создаваться уже в течение последних $2A$ единиц. Получаем, что, если N кратно трём, то ответ $\frac{N}{3} \cdot 6A + 2A = (2N + 2)A$. Если же нет, то у нас есть неполная группа в размере 1 или 2 задач. Для них ответ будет равен $\lfloor \frac{N}{3} \rfloor \cdot 6A + XA$, где X равен 6, если размер неполной равен одному, и 7, если двум.

2. Пусть $A < B < 2A$. Этот случай аналогичен случаю $B = A$ в том смысле, что Паша может работать сразу над двумя задачами. Одна пара выполняется $3A + 2B$ времени, и в последние B единиц времени мы можем начать делать следующую пару. Получается формула $\frac{N}{2} \cdot (3A + B) + B$ для чётного количества и $\frac{N+1}{2} \cdot (3A + B) - A + B$ для нечётного.

Для решения четвёртой подзадачи (**26 баллов**) нужно жадно симулировать решение задач. Каждую следующую задачу нужно начать как можно раньше. Для этого нужно поддерживать последний и предпоследний моменты времени, когда Паша освободился. Мы всегда можем начать делать задачу после последнего момента, а после предпоследнего — не всегда. Дело в том, что последний промежуток времени, когда работает Паша — это 3-й этап нескольких задач; а освобождается в предпоследний раз он после того, как закончит 1-й этап. Нужно поддерживать также начало этого промежутка, чтобы понять, сможет ли Паша уместить ещё одну задачу. В отличие от других подзадач, сложность работы этого решения составляет $O(N)$.

Для решения пятой подзадачи (**26 баллов**) нужно все случаи свести к одному. У нас всегда есть группы задач, которые мы делаем параллельно, у них 1-е этапы идут один за другим. Паша может параллельно работать над $\lfloor \frac{B}{A} \rfloor + 1$ задачами. Если в группе между 1-м этапом последней задачи и 3-м этапом первой задачи есть «зазор», то он меньше A и его ничем не заполнить.

Составим формулу. Для удобства введём обозначение $C = \lfloor \frac{B}{A} \rfloor + 1$. Одна группа задач находится на этапах 1-3 в течение $2CA + (B \ mod \ A)$ времени. Всего есть $\lfloor \frac{N}{C} \rfloor$ групп из C задач и, возможно, одна неполная группа. В неполную группу входит $N \ mod \ C$ задач, обозначим это число как D. Время выполнения неполной группы на этапах 1-3 равно $(C + D)A + (B \ mod \ A)$.

Остаётся прибавить B — время выполнения 4-го этапа последней задачи. Получаем формулу

$$\frac{N}{C} \cdot (2CA + (B \ mod \ A)) + B,$$

если N делится на C, иначе

$$\left\lfloor \frac{N}{C} \right\rfloor \cdot (2CA + (B \ mod \ A)) + (C + D)A + (B \ mod \ A) + B.$$

Всё это можно упростить до одной формулы

$$\left\lceil \frac{N}{C} \right\rceil \cdot (CA + (B \ mod \ A)) + NA + B.$$