Разборы задач 2-го дня Летней школы по программированию 2010, Севастополь

Задача A. Abacaba. Поскольку каждое заклинание S_n содержит в себе предыдущие S_i ($i < n$) в качестве префиксов, то указанное в условии S_∞ определено корректно. Легко доказать по индукции, что длина заклинания S_n равна $2^n - 1$ (длина S_0 равна $0 = 2^0 - 1$, а из того, что длина S_{n-1} равна $2^{n-1} - 1$, следует, что длина S_n равна $2^n - 1 = (2^{n-1} - 1) + 1 + (2^{n-1} - 1)$).

Поэтому впервые символ $[n]$ встретится на позиции 2^{n-1} (после префикса S_{n-1} длины $2^{n-1} - 1$). В строке S_{n+1} символ $[n]$ будет встречаться дважды – на позиции 2^{n-1} и на позиции на позиции $2^n + 2^{n-1}$. Следующая строка S_{n+2} добавит вхождения еще в двух позициях, отличающихся от предыдущих, на 2^{n+1}. Продолжая эти рассуждения, приходим к выводу, что символ $[n]$ будет встречаться в S_∞ во всех позициях, номера которых кратны 2^{n-1}, но не кратны 2^n. Отсюда следует, что номер символа, который стоит на позиции k, равен количеству нулей в конце двоичной записи числа k, увеличенному на 1.

Для обработки позиции k может потребоваться $O(\log^2 k)$ операций (может встретиться до $\log_2 k$ нулей, для обнаружения каждого из которых потребуется выполнить деление на 2, требующее $O(\log_{base} k)$ операций). Несмотря на это, на обработку l подряд идущих позиций необходимо не $O(l \log^2 k)$ операций, а $O(\log k(\log k + l \log l))$. Это объясняется тем, что на l последовательных позициях может оказаться не более одного числа кратного $2^{\lceil \log_2 l \rceil}$. (Может быть получена более точная оценка $O(\log k(\log k + l))$).

Улучшить асимптотически решение задачи вряд ли получится, но возможны некоторые оптимизации:

1. Выбор для представления чисел большего основания вида 10^b.
2. Для определения младших цифр двоичной записи делить не на 2, а сразу на 2^c. При этом вместо явного деления можно пользоваться битовыми операциями – двоичным сдвигом и побитовым and.
3. Работать не со всем числом k сразу, а лишь с остатком от его деления на $2^{\lceil \log_2 l \rceil}$, то есть с младшими $\lceil \log_2 l \rceil$ битами. Остальные биты понадобятся лишь тогда, когда эти обнулятся.
4. Удобнее использовать нумерацию позиций не с 1, а с 0. В этом случае символ будет определяться количеством младших не нулей, а единиц. В этом случае их количество свыше $\lceil \log_2 l \rceil$ можно определить в самом начале.

Задача B. 100 Великих Украинцев. Будем называть кандидатов, продвигаемых первой организацией, хорошими, а остальных – плохими.

Для того, чтобы помешать первой организации сделать K хороших кандидатов Великими, второй достаточно сделать так, чтобы в число Великих попали $M - K + 1$ плохих кандидатов. Разумнее всего, чтобы это были те из плохих кандидатов, которые уже имеют хороший рейтинг. После того, как вторая организация потратит свои деньги на увеличение рейтинга плохих, первой организации нужно будет сделать так, чтобы каждый из хороших кандидатов получил рейтинг выше, чем $(M - K + 1)$-ый по рейтингу плохой кандидат.

Таким образом, основная задача второй организации сводится к улучшению рейтинга $(M - K + 1)$-ого плохого кандидата. Сделать это можно следующим образом – увеличиваем рейтинг $(M - K + 1)$-ого до рейтинга $(M - K)$-ого. Если еще остаются деньги, то теперь будет необходимо увеличивать рейтинги обоих этих кандидатов до тех пор, пока они не достигнут $(M - K - 1)$-го, после чего нужно будет уже улучшать этих троих. Так продолжается до тех пор, пока либо не закончатся деньги, либо все $M - K + 1$ кандидатов будут иметь одинаковый рейтинг. Если уравняли всех этих кандидатов, а деньги еще остаются, то оставшиеся голоса распределяются между ними поровну.

Задача же первой организации после этого становится совсем простой. Пусть рейтинг $(M - K + 1)$-ого плохого кандидата равен v. Тогда на хороших кандидатов, у которых рейтинг выше v, мы денег не тратим, а оставшимся обеспечиваем рейтинг $v + 1$.

Следует отметить особый случай, когда $M = N$. Это означает, что все кандидаты будут Великими независимо от действий обоих организаций. Поэтому первая организация может вообще не тратить денег.

Задача C. Отрезок и квадраты. Задача является усовершенствованием задачи “Прямая и квадраты”, предложенной на Дальневосточной олимпиаде 2001 года и использованной Ф. Меньшиковым в своей книге “Олимпиадные задачи по программированию”). Фактически изменения касаются лишь ограничений на количество квадратов (в исходной постановке $N \leq 100$).

При небольшом N можно воспользоваться геометрическим подходом – найти уравнение прямой, проходящей через указанные в условии точки, а затем проверить каждый квадрат на пересечение с этой прямой. Для проверки следует подставить координаты каждой вершины квадрата в уравнение прямой. Если получится хотя бы одна вершина с неотрицательным значением и хотя бы одна с неположительным, то прямая имеет общую точку с квадратом. Сложность алгоритма – $O(N^2)$.

Способ решения задачи для больших N связан со следующим наблюдением – будем двигаться по отрезку из одного конца отрезка в другой. В новый квадрат мы переходим лишь когда пересечем либо вертикальную, либо горизонтальную линию. Но если мы пересекаем одновременно и вертикальную, и горизонтальную линии (то есть узел сетки), то тем самым мы переходим в новый квадрат, задевая при этом также и два смежных квадрата. Таким образом, количество квадратов, которые пересекает наш отрезок будет равно $1+V+H+P$, где V – количество вертикальных линий, H – количество горизонтальных линий, P – количество узлов, через которые проходит отрезок. Нетрудно видеть, что $V = N-1$, $H = \lceil E/K \rceil - \lfloor W/K \rfloor - 1$ (при условии $E > W$). (Здесь мы рассматриваем отрезок с выколотыми концевыми точками, поскольку краевые эффекты в любом случае придется рассматривать отдельно).

Займемся определением числа P. Ордината точки, в которой отрезок пересекает i-ую вертикаль ($x = K \cdot i$), будет равна $W + (E-W)/N \cdot i$. Нам нужно, чтобы эта координата была целым числом кратным K. Таким образом мы получаем уравнение

$$W + \frac{E-W}{N} \cdot i = K \cdot j,$$

(где i,j – целые числа), которое после домножения на N и переноса всех неизвестных в левую часть примет вид:

$$-(E-W) \cdot i + KN \cdot j = WN.$$

Это уравнение может быть решено с помощью обобщенного алгоритма Евклида. Нас интересует лишь количество решений, со значениями i и j в пределах от 0 до N не включая крайних значения.

Алгоритм Евклида требует порядка $O(\log N)$ операций деления. Поскольку каждое деление имеет сложность $O(\log^2 N)$, общая сложность алгоритма составит $O(\log^3 N)$.

Решение линейных диофантовых уравнений. Пусть дано уравнение $ax + by = c$, где a, b и c – заданные целые числа. Необходимо найти все такие пары целых чисел (x, y), которые удовлетворяют уравнению.

Обозначим $d = \text{НОД}(a, b)$. Если c не делится на d, то уравнение не имеет целочисленных решений (поскольку при любых целых x и y левая часть уравнения делится на d). Если же c кратно d, то существует бесконечно много решений. Пусть (x^*, y^*) – какое-либо одно из них, тогда любое другое может быть представлено в виде $(x^* + t \cdot b/d, y^* - t \cdot a/d)$ для некоторого целого t. И обратно, пара указанного вида при любом целом t является решением. Таким образом, для того, чтобы решить задачу, достаточно найти хотя бы одно решение уравнения $ax + by = d$.

Пусть (x_d, y_d) – решение уравнения $ax + by = d$. Тогда нетрудно видеть, что $(x_d \cdot c/d, y_d \cdot c/d)$ является решением исходного уравнения. Для решения же уравнения $ax + by = d$ используется обобщенный (или расширенный) алгоритм Евклида.

Обычный алгоритм Евклида предназначен для нахождения наибольшего общего делителя двух целых чисел a и b:

```
m1=a; m2=b;
while (m2) {
  q = m1 / m2;
  m1 -= q*m2;
  swap(m1,m2);
}
```

По окончанию алгоритма $m1$ станет равным значению $\text{НОД}(a, b)$, а $m2$ – нулю.

В обобщенном алгоритме Евклида добавляются переменные $x1$, $y1$ и $x2$, $y2$ и обеспечивается, что при всех изменениях $m1$ и $m2$ сохраняются равенства $a \cdot x1 + b \cdot y1 = m1$ и $a \cdot x2 + b \cdot y2 = m2$.

```
x1=1; y1=0; m1=a;
x2=0; y2=1; m2=b;
while (m2) {
  q = m1 / m2;
```

```
    (x1,y1,m1) -= q*(x2,y2,m2);
    swap( (x1,y1,m1) , (x2,y2,m2) );
  }
```

Как и прежде, по окончанию алгоритма $m1$ станет равным значению $d = \text{НОД}(a, b)$, а значит $(x1, y1)$ будет решением уравнения $ax + by = d$. Если ни одно из чисел a и b не равно 0, то $|x1| < |b/d|$, $|y1| < |a/d|$. Кроме того, полезным побочным результатом является то, что $x2 = \pm b/d$, а $y2 = \mp a/d$. Этих значений оказывается достаточно чтобы записать общее решение уравнения $ax + by = c$.

Задача D. Обмен между тремя героями. Пусть a_1, a_2, a_3 – количество воинов в армиях первого, второго и третьего героев соответственно. Без ограничения общности можно считать, что $a_1 \le a_2 \le a_3$ (если это не так, их можно переупорядочить).

Заметим, что в случае, если по крайней мере у двух героев нет армий ($a_1 = a_2 = 0$), то невозможно будет произвести ни одного хода. Если только один из героев не имеет армии ($a_1 = 0$, $a_2 \neq 0$), то обмены будут происходит между двумя другими героями и очевидно ходов не может быть больше, чем количество воинов в наименьшей из армий (a_2). Все случаи с нулевыми значениями мы рассмотрели, поэтому в дальнейшем будем предполагать, что все герои имеют непустые армии.

Оценим сверху количество возможных ходов. Каждый воин может переместиться не более двух раз. А поскольку каждый ход связан с двумя перемещениями, общее число ходов не может превысить общего количества воинов во всех армиях (то есть $a_1 + a_2 + a_3$).

Рассмотрим случай, когда $a_1 + a_2 < a_3$. Тогда количество ходов не изменится, если a_3 заменить суммой $a_1 + a_2$. Действительно, после использования $a_1 + a_2$ воинов армии третьего героя армии первого и второго героев будут состоять из воинов, которые изначально были в армии третьего героя (ведь возвращаться они не могут). Поэтому оставшаяся армия третьего героя не может участвовать в обменах.

Перейдем теперь к случаю $a_1 + a_2 = a_3$. Рассмотрим пример $a_1 = 1$, $a_2 = k$, $a_3 = k + 1$. Нетрудно видеть, что указанная ранее оценка числа ходов достигается, если использовать следующую последовательность действий:

- обмен $1 \longleftrightarrow 2$ приводит к состоянию $\{1(2)\}$, $\{2(1),\ k-1$ раз $2\}$, $\{k+1$ раз $3\}$;
- повторить для i от 1 до $k-1$ раз следующие действия:
  - обмен $1(2) \longleftrightarrow 3$ приводит к состоянию $\{1(3)\}$, $\{i-1$ раз $2(13),\ 2(1),\ k-i$ раз $2\}$, $\{i$ раз $3(12),\ k-i+1$ раз $3\}$;
  - обмен $1(3) \longleftrightarrow 2$ приводит к состоянию $\{1(2)\}$, $\{i$ раз $2(13),\ 2(1),\ k-i-1$ раз $2\}$, $\{i$ раз $3(12),\ k-i+1$ раз $3\}$;
- обмен $1(2) \longleftrightarrow 3$ приводит к состоянию $\{1(3)\}$, $\{k-1$ раз $2(13),\ 2(1)\}$, $\{k$ раз $3(12),\ 3\}$;
- обмен $2(1) \longleftrightarrow 3$ приводит к состоянию $\{1(3)\}$, $\{k-1$ раз $2(13),\ 2(3)\}$, $\{k+1$ раз $3(12)\}$;
- обмен $1(3) \longleftrightarrow 2(3)$ приводит к состоянию $\{1(23)\}$, $\{k$ раз $2(13)\}$, $\{k+1$ раз $3(12)\}$.

Здесь и далее воин обозначается номером героя, в армии которого он находится, а в скобках указываются номера героев, в армиях которых он уже побывал.

Любую конфигурацию вида $(a_1, a_2, a_1 + a_2)$ можно разбита на конфигурацию $(1, a_2 - a_1 + 1, a_2 - a_1 + 2)$ и $a_1 - 1$ конфигураций $(1, 1, 2)$, для которых обмены можно производить по приведенной выше схеме. Это доказывает, что оценка $a_1 + a_2 + a_3$ достигается для любой такой конфигурации.

Для конфигураций, где $a_1 + a_2 > a_3$, можно за счет нескольких конфигураций $(2, 2, 2)$ (их будет $\lfloor (a_1 + a_2 - a_3)/2 \rfloor$), и возможно одной $(1, 1, 1)$ (в случае, если понадобилось округление вниз при делении на 2), прийти к конфигурации вида $a_1 + a_2 = a_3$, однако следует заметить, что если для $(2, 2, 2)$ достигается оценка 6, то для $(1, 1, 1)$ максимум чего мы можем добиться – это 2.

Для того, чтобы быть уверенным в том, что указанное разбиение дает оптимальное количество обменов, следует показать, что в случае, когда сумма $S = a_1 + a_2 + a_3$ нечетна, невозможно выполнить S операций обменов. Назовем перемещения войск из 1 в 2, из 2 в 3 и из 3 в 1 перемещениями в прямом направлении, а из 1 в 3, из 2 в 1 и из 3 в 2 – в обратном. Очевидно, что войска, продвинувшиеся на один шаг в каком-либо

направлении, следующий свой ход обязаны делать в том же направлении. Рассмотрим войска, успевшие сделать лишь один ход в прямом направлении (то есть войска типа 1(2), 2(3) и 3(1)). Поскольку каждый обмен перемещает одно войско в прямом направлении (а другое в обратном), то количество рассматриваемых войск может либо увеличиться на 1, либо уменьшиться на 1, то есть изменяется честность этого количества. В единственно возможном после S обменов состоянии (когда все войска побывали по одному разу у каждого героя) войск, продвинувшихся на один шаг вперед, нет. Также нет их и в начальном состоянии. В силу изменения четности, мы не можем попасть из одного в другое за нечетное количество ходов S. В то же время, как было показано выше, можно выполнить $S-1$ обменов. Поэтому это и будет оптимальным числом обменов при нечетном общем количестве войск.

Задача Е. Упражнение по арифметике. При очень малых N задача может быть решена прямым перебором. Для этого, для каждого N, начиная с 1, нужно проверить, используя все возможные расстановки знаков в примере, может ли ответ быть равным k. Такое решение будет иметь сложность $O(N \cdot 2^N)$, где N – результат. Очевидно, что это очень много. Кроме того, совершенно не очевидно, что такое N будет найдено и наш алгоритм завершится. Тем не менее, дальше будет показано, что этот алгоритм действительно находит нужное значение.

Прежде всего, отметим, что ответ задачи для значений k и $-k$ будет одинаковым. Действительно, если существует расстановка знаков в выражении ? 1 ? 2 ? ... ? N, которая дает результат k, то изменив в нем все знаки на противоположные, получим значение $-k$. Таким образом, взяв модуль, можно свести задачу к ограничениям $k \geq 0$. Далее, совершенно ясно, что при больших значениях k, результат N не может быть маленьким. Действительно, при любой расстановке знаков выражение ? 1 ? 2 ? ... ? N не превосходит $+1 + 2 + \cdots + N = N(N+1)/2$. Поэтому, все значения, для которых $N(N+1)/2$ меньше k заведомо не удовлетворяют условию задачи. Решим неравенство

$$\frac{N(N+1)}{2} \geq k.$$

Умножим обе части на 2 и перенесем все в левую часть:

$$N^2 + N - 2k \geq 0.$$

Это квадратное неравенство имеет решение

$$\left[\begin{array}{l} N \leq -\frac{1}{2} - \sqrt{\frac{1}{4} + 2k}, \\ N \geq -\frac{1}{2} + \sqrt{\frac{1}{4} + 2k}. \end{array}\right.$$

Поскольку N не может быть отрицательным и дробным, то

$$N \geq \left\lceil -\frac{1}{2} + \sqrt{\frac{1}{4} + 2k} \right\rceil.$$

Чтобы получить решение со сложностью $O(1)$ попробуем проанализировать какие числа k могут получиться при фиксированном значении N:

N	k
1	-1 1
2	-3 -1 1 3
3	-6 -4 -2 0 2 4 6
4	-10 -8 -4 -2 0 2 4 6 8 10
$\vdots$	$\vdots$

Вычисленные значения наводят на мысль о том, что при заданном значении N могут получиться все значения от $-N(N+1)/2$ до $N(N+1)/2$ с шагом 2. Докажем это.

Прежде всего заметим, что числа другой четности получить невозможно. Это действительно так, потому что числа i и $-i$ имеют одинаковую четность. А значит остаток от деления выражения ? 1 ? 2 ? ... ? N на 2 не зависит от расстановки знаков.

Остается лишь показать, что все числа той же четности, что и $N(N+1)/2$ могут быть получены. Как видно из таблицы, для первых четырех значений N это выполняется. Пусть данное утверждение верно для $N-1$. То есть для каждого k', не превосходящего по модулю значения $(N-1)N/2$ и имеющего ту же четность, есть некоторая расстановка знаков в выражении ? 1 ? 2 ? ... ? $(N-1)$, которая делает его равным k'. Возьмем $k \in [-N(N+1)/2, N(N+1)/2]$ такое, что $k \equiv N(N+1)/2 \pmod 2$. Тогда, хотя бы одно из чисел $k-N$ и $k+N$ принадлежит отрезку $[-N(N-1)/2, N(N-1)/2]$ (при $k \geq -(N-1)N/2$ это число $k-N$, а при $k \leq (N-1)N/2$ – число $k+N$). Кроме того, $k \pm N \equiv N(N+1)/2 \pm N \equiv N(N-1)/2 (\mathrm{mod}\ 2)$. Поэтому, по предположению индукции, хотя бы одно из $k-N$ и $k+N$ может быть представлено в виде ? 1 ? 2 ? ... ? $(N-1)$. Добавляя (или вычитая в случае $k+N$) к полученному выражению N, мы получим представление числа k в виде ? 1 ? 2 ? ... ? N.

Таким образом, мы находим среди всех чисел N, не меньших $\left\lceil -\frac{1}{2} + \sqrt{\frac{1}{4} + 2k} \right\rceil$, первое такое, что $k \equiv N(N+1)/2 \pmod 2$). Для этого, придется сделать не более двух увеличений N. И поэтому алгоритм будет иметь сложность $O(1)$.

Этот факт объясняется тем, что четность числа $N(N+1)/2$ зависит лишь от $N \bmod 4$. Действительно, если $N = 4q + r$.

$$\frac{N(N+1)}{2} = \frac{(4q+r)(4q+r+1)}{2} = 8q^2 + 2q(2r+1) + \frac{r(r+1)}{2}.$$

Первые два слагаемых делятся на 2, поэтому четность суммы совпадает с четностью третьего члена – $r(r+1)/2$. При $r = 0$ или $r = 3$ мы имеем четную сумму, а при $r = 1$ или $r = 2$ нечетную. Таким образом, при несовпадении четности k и $N(N+1)/2$ следует увеличить N до ближайшего большего нечетного.

Следует отметить особый случай $k = 0$. В этом случае ответ, который даст наш алгоритм – $N = 0$. Однако в условии сказано, что N должно быть натуральным числом, поэтому ответ должен быть $N = 3$.

Сложность алгоритма определяется сложностью извлечения квадратного корня и составляет $O(\log^2 k)$.

Задача F. Ремейки. Для эффективного решения задачи достаточно заметить, что у мотива и транспонированного мотива совпадают интервалы (разности) между последовательными нотами. Тогда, если мы преобразуем мелодию и мотив к разностному виду, задача сведется к проверке точного вхождения мотива в мелодию. Например, для второго теста из примера входных данных мотив $(4, 5, 7)$ преобразуется к виду $(1, 2)$, а мелодия $(-1, -2, -1, 1, 4, 5, 8)$ – к $(-1, 1, 2, 3, 1, 3)$. Отсюда видно, что мотив входит в мелодию, начиная со второй позиции.

Остается лишь применить какой-либо из эффективных алгоритмов поиска подстроки в строке. Например, алгоритм Кнута-Мориса-Пратта или Бойера-Мура. Вычислительная сложность – $O(M+N)$.

Отдельно следует рассмотреть случаи, когда мотив состоит из одной ноты и когда длина мелодии меньше длины мотива.

Алгоритм Кнута-Мориса-Пратта. Пусть задана строка $P = p_1 p_2 \dots p_M$ (образец) и строка $T = t_1 t_2 \dots t_N$ (текст). Число s называется допустимым сдвигом, если $0 \leq s \leq N - M$ и $T[s+j] = P[j]$ для всех $j = \overline{1, M}$. Задача поиска подстрок заключается в нахождении всех допустимых сдвигов для заданных текста и образца.

Конкатенацией xy двух строк x и y называется строка, получающаяся при выписывании строки x, а следом за ней встык строки y. Строка w называется префиксом строки x ($w \sqsubset x$), если $x = wy$ для некоторой строки y. Строка w называется суффиксом строки x ($w \sqsupset x$), если $x = yw$ для некоторой строки y.

Обозначим P_k – префикс строки P, состоящий из первых k символов (при этом P_0 будет пустой строкой, а $P_M = P$). Тогда сдвиг s будет допустимым, если $P \sqsupset T_{s+M}$.

Построим для строки P префикс-функцию $\pi : \overline{1, M} \to \overline{0, M-1}$, определяемую следующим образом:

$$\pi[q] = \max\{k : k < q \text{ и } P_k \sqsupset P_q\}.$$

Имея такую функцию, нам не понадобится при несовпадении очередного символа возвращаться назад для проверки следующего сдвига на допустимость. Имея информацию о том, что $P_q \sqsupset T_i$, мы знаем так же и о том, что и $P_{\pi[q]} \sqsupset T_i$, при этом ни для какого k, удовлетворяющего $\pi[q] < k < q$, P_k не будет суффиксом строки T_i. Поэтому мы можем пропустить проверку некоторых сдвигов как заведомо недопустимых, а тот, который может оказаться допустимым, проверять не с самого начала.

Алгоритм может быть записан следующим образом:

```
q = 0;
for (i=1; i<=N; i++)
{
  while (q && P[q+1]!=T[i])
      q = pi[q];
  if (P[q+1]==T[i])
  {
      ++q;
      if (q==M)
      {
          P входит в T со сдвигом i-M
          q = pi[q];
      }
  }
}
```

Нетрудно понять, что количество действий, выполняемых этим алгоритмом, можно оценить как $O(N)$. Определенные сомнения может вызывать цикл while. Однако достаточно заметить, что каждая итерация этого цикла уменьшает q по крайней мере на 1 (поскольку по определению $\pi[q] < q$). Оценка же $O(N)$ следует из того, что увеличиваться q может не более N раз.

Построение же префикс-функции происходит с помощью аналогичного кода:

```
pi[1] = q = 0;
for (i=2; i<=M; i++)
{
  while (q && P[q+1]!=P[i])
      q = pi[q];
  if (P[q+1]==P[i])
      ++q;
  pi[i]=q;
}
```

Сложность этой части алгоритма составляет $O(M)$. Это может быть доказано теми же рассуждениями, что и в предыдущем случае.

Задача F. Определитель. Определитель квадратной матрицы будем находить методом Гаусса, который состоит в приведении матрицы к верхнетреугольному виду с помощью элементарных преобразований, которые не меняют определитель, или меняют его знак. Тогда определитель будет, очевидно, равен произведению диагональных элементов. Суть метода состоит в следующем. На шаге i наша матрица будет иметь вид

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,i-1} & a_{1,i} & a_{1,i+1} & \dots & a_{1,n} \\ 0 & a_{2,2} & \dots & a_{2,i-1} & a_{2,i} & a_{2,i+1} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{i-1,i-1} & a_{i-1,i} & a_{i-1,i+1} & \dots & a_{i-1,n} \\ 0 & 0 & \dots & 0 & a_{i,i} & a_{i,i+1} & \dots & a_{i,n} \\ 0 & 0 & \dots & 0 & a_{i+1,i} & a_{i+1,i+1} & \dots & a_{i+1,n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & a_{n,i} & a_{n,i+1} & \dots & a_{n,n} \end{pmatrix}.$$

Цель – обнулить элементы i-того столбца под главной диагональю. Для этого находим среди элементов $a_{i,i}, a_{i+1,i}, \dots, a_{n,i}$ ненулевой элемент. Если все они нули, то можно сразу сказать, что определитель равен нулю и закончить процесс. Если же нашелся ненулевой элемент $a_{j,i}$, то если $j \neq i$, то мы меняем i-тую и j-тую строки матрицы, помечая при этом, что знак определителя поменялся на противоположный. После этого мы для каждого $j > i$ вычитаем из j-той строки i-тую, домноженную на $a_{j,i}/a_{i,i}$. Это преобразование сохраняет

определитель. Будем называть его "чисткой", а одно действие в "чистке" – атомарной операцией "чистки". Таким образом, при переходе к следующему шагу матрица будет иметь нужный вид. Дойдя до n-ого шага, мы приведем матрицу к требуемому виду и найдем ее определитель с учетом информации о перемене знака. Легко оценить, что общее число атомарных операций "чистки" в методе Гаусса не превосходит $2n^3/3+O(n^2)$, и ясно, что каждая из них выполняется за $O(1)$.

Оценим максимальный ответ при ограничениях задачи. По определению определителя как суммы $n!$ произведений по n чисел имеем, что он по модулю не превосходит $10^{6n}n! \le 10^{600}100! \le 10^{758}$. Можно получить более точную оценку. По неравенству Адамара $\det(a_{i,j})_{i,j=1}^n \le \prod_{i=1}^n \sqrt{a_{i,1}^2 + a_{i,2}^2 + \ldots + a_{i,n}^2} \le \left(10^6\sqrt{n}\right)^n \le 10^{700}$. Причем в тестах есть матрица, определитель которой равен примерно $5 \cdot 10^{692}$, что показывает довольно неплохую точность данной оценки. Следует также отметить, что эта оценка достигается, когда n степень двойки. Из этого следует, что проводить вычисления в каких-то стандартных типах не представляется возможным.

Один из способов решить проблему – проводить вычисления, используя длинную арифметику (либо длинное рациональное число, либо длинное действительное число, можно даже оставаться в целом типе, видоизменив немного сам метод). Но тогда к шагу, скажем, с номером $n/2$ в оставшейся части матрицы могут быть очень большие числа (порядка 350 знаков), и начнет сказываться вычислительная сложность операции умножения длинных чисел: на каждую атомарную операцию "чистки" будет еще уходить $O(n^2)$ операций, связанных с длинными числами. Поэтому алгоритм будет иметь сложность $O(n^5)$ (и даже $O(n^6)$, если работа идет с рациональными или целыми числами). Подобные алгоритмы требуют слишком много времени, поэтому необходим какой-то другой подход.

Заметим, что вычислять определитель по простому модулю p можно тем же самым методом с одной маленькой заменой: домножающий коэффициент в операции "чистки" равен не $a_{j,i}/a_{i,i}$, а $a_{j,i} \cdot \mathrm{inv}_p(a_{i,i})$, где $\mathrm{inv}_p(x)$ удовлетворяет свойству $x \cdot \mathrm{inv}_p(x) \equiv 1 \pmod p$. $\mathrm{inv}_p(x)$ можно, например, найти по формуле $\mathrm{inv}_p(x) = x^{p-2} \bmod p$ с помощью двоичного алгоритма возведения в степень за $O(\log p)$, которая справедлива в силу малой теоремы Ферма. Но мы можем вычислить $\mathrm{inv}_p(a_{i,i})$ один раз перед всеми операциями "чистки", и находить в каждом случае домножающий коэффициент за $O(1)$.

Далее заметим, что если нам надо найти какое-то целое число и мы вычислили его по достаточно большому числу различных простых модулей (а именно, их произведение гарантировано более чем в два раза больше модуля этого числа), то по китайской теореме об остатках мы можем восстановить это число однозначно даже со знаком. Действительно, если восстановленное число больше половины произведения этих модулей, то очевидно оно должно быть отрицательным и надо просто вычесть из него это произведение. Таким способом наиболее целесообразно находить определитель.

В данной задаче наиболее удобно брать простые числа следующие за 10^9 (намного больше брать нельзя, так как в этом случае невозможно будет проводить все операции в стандартных целочисленных типах). Так как определитель гарантировано не превосходит по модулю 10^{700}, то достаточно взять 80 первых простых чисел, превосходящих 10^9: их произведение будет больше, чем $(10^9)^{80} = 10^{720}$, что более чем в два раза превышает определитель по модулю. Вообще количество этих простых чисел надо брать большим, чем $\log_{10^9}\left((10^6\sqrt{n})^n\right) = n(12+\lg n)/18 = O(n\log n)$. Итак, мы выбрали $k = 80$ различных простых чисел $p_1, p_2, \ldots, p_k$, вычислили определитель по модулю каждого из них, получили числа $r_1, r_2, \ldots, r_k$. На все это ушло около $2n^3/3 \cdot 80 \le 67 \cdot 10^6$ атомарных операций "чистки", что вполне приемлемо. Сама же сложность алгоритма равна $O(n^4 \log n)$, но с очень маленькой константой.

Теперь нам надо восстановить сам определитель d, исходя из условий, что $d \equiv r_i \pmod{p_i}$ $(1 \le i \le k)$. Будем искать его в виде $d = v_1 + p_1v_2 + p_1p_2v_3 + \ldots + p_1\ldots p_{k-1}v_k$, где $0 \le v_i < p_i$. Тогда ясно, что $v_1 = r_1$. На v_2 мы имеем условие $v_1 + p_1v_2 \equiv r_2 \pmod{p_2}$. Поэтому $v_2 = (r_2 - v_1)\mathrm{inv}_{p_2}(p_1) \bmod p_2$. Пусть мы уже нашли числа $v_1, \ldots, v_{i-1}$. Тогда на v_i мы получаем сравнение $v_1 + p_1v_2 + \ldots + p_1\ldots p_{i-1}v_i \equiv r_i \pmod{p_i}$. Проводя последовательно преобразования, получим сравнения

$$v_2 + p_2v_3 + \ldots + p_2\ldots p_{i-1}v_i \equiv (r_i - v_1)\mathrm{inv}_{p_i}(p_1) \pmod{p_i};$$

$$v_3 + p_3v_4 + \ldots + p_3\ldots p_{i-1}v_i \equiv ((r_i - v_1)\mathrm{inv}_{p_i}(p_1) - v_2)\mathrm{inv}_{p_i}(p_2) \pmod{p_i};$$

$$\ldots$$

$$v_i \equiv (\ldots((r_i - v_1)\mathrm{inv}_{p_i}(p_1) - v_2)\mathrm{inv}_{p_i}(p_2) - \ldots - v_{i-1})\mathrm{inv}_{p_i}(p_{i-1}) \pmod{p_i}.$$

Таким образом, мы можем найти числа $v_1, v_2, \ldots, v_k$ за $O(k^2 \log p)$, где $p = \max\{p_i : 1 \le i \le k\}$, оставаясь при этом в стандартных целочисленных типах. Теперь используя довольно простые операции с длинными числами (сложение и умножение на короткое), мы можем найти d по формуле $d = (\ldots((v_k p_{k-1} + v_{k-1})p_{k-2} + v_{k-2}) + \ldots + v_2)p_1 + v_1$ (она напоминает схему Горнера). Если $2d > P$, где $P = p_1 \ldots p_k$, то ответ равен $d - P$, иначе он равен d. P можно считать одновременно с d. Получается, что в конце нам нужны еще операции сравнения и вычитания длинных чисел. Таким образом, из операций с длинными числами понадобились лишь базовые.

Задача G. Сумма произведений. Обозначим $p = 1000000009$ и $d = b - a$. Ясно, что ответ не изменится, если мы уменьшим оба числа a и b на одно и то же число кратное p. Поэтому мы можем заменить a на $a \bmod p$, а b на $a + d$. Откуда $a \le b = a + d < p + 10^7 = 10^9 + 10^7 + 9$, так как по условию $d \le 10^7$.

Наиболее простая идея здесь – это поддерживать текущее произведение $i \cdot (i+1) \cdot \ldots \cdot (i+n-1)$ и накапливать сумму таких произведений в переменную с ответом. Тогда при $i = a$ его надо вычислить непосредственно, после чего при переходе от $i-1$ к i надо домножить его на $i+n-1$ и поделить на $i-1$. Но проблема в том, что делить надо по модулю, а для этого надо находить обратный остаток $\mathrm{inv}_p(i-1)$, где $\mathrm{inv}_p(x)$ удовлетворяет свойству $x \cdot \mathrm{inv}_p(x) \equiv 1 \pmod p$. Один из способов вычисления $\mathrm{inv}_p(x)$ – использовать формулу $\mathrm{inv}_p(x) = x^{p-2} \bmod p$, которая справедлива в силу малой теоремы Ферма, и двоичный алгоритм возведения в степень за $O(\log p)$. Кроме того, $i-1$ может оказаться кратным p, и тогда на него вообще нельзя делить. Но эта проблема обходится легко. Число кратное p среди чисел от a до $b-1$ может быть только одно, либо 0, либо p. Все произведения, в которые оно входит, равны нулю по модулю p. Поэтому в этом случае мы можем просто решить задачу независимо для двух отрезков, где уже не придется делить на p, и ответы сложить. Таким образом, пока мы получаем алгоритм со сложностью $O(n + d \log p)$, причем с большой скрытой константой, так как формула для обратного остатка выше требует проводить вычисления в 64-битовом целочисленном типе. Эту константу можно уменьшить, если использовать расширенный алгоритм Евклида, тогда обратный остаток будет вычисляться в рамках 32-битного типа, что уже значительно ускорит процесс, кроме того суммарное число итераций алгоритма Евклида для вычисления всех обратных остатков от a до $b-1$, будет значительно меньше, чем его теоретическая оценка. Но при таких ограничениях на n и d даже такой алгоритм вряд ли уложится по времени. Будем называть этот алгоритм "алгоритм Евклида".

Теперь опишем довольно простой алгоритм со сложностью $O(n+d)$, и требующий $4\min\{n,d\} + O(1)$ бит памяти. Предположим пока, что $d \le n$. Вычислим сначала величины $\mathrm{pref}[i] = i \cdot (i+1) \cdot \ldots \cdot (b-1)$ для всех i от a до b ($\mathrm{pref}[i] = 1$ при $i = b$). Это можно сделать по рекуррентной формуле $\mathrm{pref}[i] = i \cdot \mathrm{pref}[i+1]$. Затем будем поддерживать произведение вида $b \cdot (b+1) \cdot \ldots \cdot j$ в переменной suf, где j пробегает значения от b до $b+n-1$. Пока $j < a+n-1$ мы просто поддерживаем величину suf. Тут как раз важно, что $b \le a+n$. При каждом j от $a+n-1$ до $b+n-1$ произведение $\mathrm{pref}[j-n+1] \cdot \mathrm{suf}$ будет равно $(j-n+1) \cdot (j-n+2) \cdot \ldots \cdot j$, то есть слагаемому интересующей нас суммы, поэтому мы можем за один проход найти нашу сумму. При этом одна итерация цикла будет занимать $O(1)$. Если же $d > n$, то надо просто разбить сумму на несколько сумм, в каждой из которых $d \le n$, вычислить каждую сумму по выше описанному алгоритму и ответы сложить. Оценим более детально число операций умножения в этом алгоритме (которые должны будут проходить в 64-битовом целом типе). Когда $d \le n$, то оно составляется из $d+1$ умножений вида $\mathrm{pref}[j-n+1] \cdot \mathrm{suf}$, из d умножений для нахождения величин $\mathrm{pref}[i]$ и еще из n умножений для поддержки переменной suf. Всего $n+2d+1$ умножений. Когда $d > n$ и мы разбили отрезок от a до b на некоторое число отрезков, для которых $d \le n$, то общее число умножений будет равно $\sum_{i=1}^{k}(n + 2d_i + 1) = 2d + 2 + k(n-1)$, так как $\sum_{i=1}^{k}(d_i + 1) = d + 1$. Так как $d_i \le n$, то $d+1 = \sum_{i=1}^{k}(d_i+1) \le k(n+1)$, то есть $k \ge \frac{d+1}{n+1}$. Причем ясно, что для $k = \lceil \frac{d+1}{n+1} \rceil$ такое разбиение существует. Значит, общее число операций умножения равно $2d+2+\lceil \frac{d+1}{n+1} \rceil (n-1) \le 2d+2+\frac{d+n+1}{n+1}(n-1) = 3d+n+1-\frac{2d}{n+1}$. Причем эта оценка достигается, например, при $d = n+1$. То есть наиболее плохим тестом для этого алгоритма будет тест, где $d = 10^7$, а $n = 10^7 - 1$, для которого будет произведено почти 40 миллионов операций умножения, что тем не менее вполне приемлемо по времени.

Третий подход состоит в применении дерева отрезков для операции произведения, построенного на числах от a до $b+n-1$. Сначала за $O(d+n)$ мы строим дерево отрезков, а потом вычисляем искомую сумму за d запросов в дерево снизу, каждый за $O(\log n)$. Будем называть этот алгоритм "дерево отрезков". Сложность этого алгоритма равна $O(n + d + d \log n)$, что на вид немного лучше, чем "алгоритм Евклида", (ведь n значительно меньше p), но как показало непосредственное сравнение, "дерево отрезков" проигрывает раза в два по времени "алгоритму Евклида". Это связано с тем, что в "дереве отрезков" $O(d \log n)$ умножений

в 64-битном типе, а в "алгоритме Евклида" $O(d \log p)$ умножений и делений в 32-битном типе. С помощью различных приемов можно ускорить "дерево отрезков", но не более чем до "алгоритма Евклида". Сам же "алгоритм Евклида", проигрывает второму подходу где-то в 4 раза.

Все эти подходы используют ограничение $b - a \le 10^7$, причем по сути они годятся для вычисления сумм вида $a[1] \cdot a[2] \cdot \ldots \cdot a[n] + a[2] \cdot a[3] \cdot \ldots \cdot a[n+1] + \ldots + a[d+1] \cdot a[d+2] \cdot \ldots \cdot a[d+n]$ для любых массивов чисел $a[1], a[2], \ldots, a[d+n]$.

Но конкретно для этой суммы существует еще один подход со сложностью $O(n)$, который никак не использует близость чисел a и b. Он основан на формуле

$$\sum_{i=1}^{k} i \cdot (i+1) \cdot \ldots \cdot (i+n-1) = \frac{k \cdot (k+1) \cdot \ldots \cdot (k+n-1) \cdot (k+n)}{n+1},$$

которую можно легко доказать по индукции.

Тогда искомая сумма от a до b равна разности

$$\frac{b \cdot (b+1) \cdot \ldots \cdot (b+n) - (a-1) \cdot a \cdot \ldots \cdot (a+n-1)}{n+1},$$

которая легко может вычислена за $2n$ умножений и одно деление по модулю p, которое возможно, так как $n \le 10^7$, а $p > 10^9$.

Задача H. Сумма различных делителей. Пусть $N = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$ – разложение N на простые множители, причем $p_1 < p_2 < \ldots < p_k$. Введем для N следующие промежуточные делители: $d_0 = 1$, $d_1 = p_1^{a_1}$, $d_2 = p_1^{a_1} p_2^{a_2}$, …, $d_k = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k} = N$ и для удобства еще обозначим $p_{k+1} = \infty$. Сумму делителей натурального числа n будем обозначать $\sigma(n)$.

Главным для решения задачи является следующее утверждение.

Теорема. Пусть выполнены неравенства $p_1 \le \sigma(d_0)+1, p_2 \le \sigma(d_1)+1, \ldots, p_s \le \sigma(d_{s-1})+1$, но $p_{s+1} > \sigma(d_s)+1$. (Так как $p_{k+1} = \infty$, то такое s всегда найдется.) Тогда наименьшее натуральное число не представимое в виде суммы различных делителей числа N равно $\sigma(d_s) + 1$.

Доказательство. Докажем по индукции, что при $0 \le i \le s$ все целые числа от 0 до $\sigma(d_i)$ представимы в виде суммы (возможно пустой) различных натуральных делителей числа d_i. База при $i = 0$ очевидна. Пусть утверждение доказано для $i = j - 1$ и докажем его при $i = j$. Обозначим $S = \sigma(d_j)$, $s = \sigma(d_{j-1})$, $p = p_j$ и $a = a_j$. Тогда ясно, что $S = s \cdot (1 + p + p^2 + \ldots + p^a)$. Нам дано, что любое целое число от 0 до s представимо в виде суммы (возможно пустой) различных натуральных делителей числа d_{j-1}, а надо доказать, что любое целое число от 0 до S представимо в виде суммы (возможно пустой) различных натуральных делителей числа d_j Все делители числа d_j имеют вид $d' p^{a'}$, где d' – произвольный делитель d_{j-1} и $0 \le a' \le a$. Теперь мы опять применим индукцию, чтобы доказать, что для любого b от 0 до a верно, что любое целое число от 0 до $s \cdot (1 + p + \ldots + p^b)$ представимо в виде суммы (возможно пустой) различных натуральных делителей числа $d_{j-1} p^b$. База при $b = 0$ очевидна – это просто предположение первой индукции. Пусть для $b = c - 1$ утверждение доказано, докажем его для $b = c$. Пусть $A \in \{0, 1, \ldots, s \cdot (1 + p + \ldots + p^{c-1})\}$ и $B \in \{0, 1, \ldots, s\}$. Тогда по предположению индукции $A = \sum_u x_u$, где x_u какие-то различные делители числа $d_{j-1} p^{c-1}$, и $B = \sum_v y_v$, где y_v какие-то различные делители числа d_{j-1}, так как $d_{j-1} p^{c-1}$ не делится на p^c, то $x_u \ne y_v p^c$ при всех u и v. Поэтому равенство $C = A + B p^c = \sum_u x_u + \sum_v y_v p^c$, дает представление числа C в виде суммы различных натуральных делителей числа $d_{j-1} p^c$. Далее заметим, что по условию $p = p_j \le \sigma(d_{j-1}) + 1 = s + 1$. Поэтому $p^c = (p-1)(1 + p + \ldots + p^{c-1}) + 1 \le s \cdot (1 + p + \ldots + p^{c-1}) + 1$. Откуда следует, что числа вида $A + B p^c$ пробегают всё множество $\{0, 1, \ldots, s \cdot (1 + p + \ldots + p^c)\}$, когда A пробегает множество $\{0, 1, \ldots, s \cdot (1 + p + \ldots + p^{c-1})\}$, а B – множество $\{0, 1, \ldots, s\}$. Таким образом, утверждение второй индукции доказано, а, значит, и утверждение первой индукции. Следовательно, все числа от 0 до $\sigma(d_s)$ представимы в виде суммы различных натуральных делителей числа N.

Рассмотрим теперь число $\sigma(d_s)+1$. Допустим, что оно представимо в виде суммы различных натуральных делителей числа N. Сумма всех делителей, которые не делятся ни на одно из простых чисел $p_{s+1}, \ldots, p_k$ равна $\sigma(d_s)$, так как это в точности делители числа d_s. Значит, в представлении числа $\sigma(d_s) + 1$ в виде суммы различных натуральных делителей числа N есть хотя бы один делитель, кратный одному из чисел $p_{s+1}, \ldots, p_k$. При $s = k$ это дает очевидное противоречие. Если же $s < k$, то в силу неравенства $p_{s+1} > \sigma(d_s)+1$ этот делитель заведомо больше $\sigma(d_s) + 1$ и мы опять получаем противоречие.

Таким образом, $\sigma(d_s)+1$ – это наименьшее натуральное число не представимое в виде суммы различных делителей числа N, что и требовалось доказать.

Эта теорема дает простой алгоритм нахождения ответа на задачу. Будем раскладывать число N на простые множители обычным способом за $O(\sqrt{N})$, при этом будем поддерживать текущий делитель d_j и его сумму делителей $\sigma(d_j)$. Тогда при поиске очередного простого делителя N мы должны кроме неравенства $p^2 \le N/d$ также проверять неравенство $p \le s+1$, где d – текущий делитель d_j, а s – его сумма делителей, и в любом из этих случаев выходить из цикла, после чего очевидным образом обрабатывать оставшийся простой делитель, если он есть. Заметим, что из условий выхода из цикла следует, что пока цикл выполняется, выполнено неравенство $p^3 \le N\frac{\sigma(d)+1}{d}$. Оценим величину $\frac{\sigma(n)}{n}$ при $n \le 2\cdot 10^{18}$. Заметим, что если

$$n = p_1^{a_1} \dots p_k^{a_k},$$

то

$$\frac{\sigma(n)}{n} = \left(1+\frac{1}{p_1}+\ldots+\frac{1}{p_1^{a_1}}\right)\dots\left(1+\frac{1}{p_k}+\ldots+\frac{1}{p_k^{a_k}}\right)$$
$$\le \left(1+\frac{1}{p_1}+\ldots+\frac{1}{p_1^{a_1}}+\ldots\right)\dots\left(1+\frac{1}{p_k}+\ldots+\frac{1}{p_k^{a_k}}+\ldots\right) = \frac{p_1}{p_1-1}\dots\frac{p_k}{p_k-1}$$
$$\le \frac{q_1}{q_1-1}\dots\frac{q_k}{q_k-1},$$

где $q_1=2$, $q_2=3$, $q_3=5$, ..., то есть $q_1, q_2, \dots$ – это последовательность всех простых чисел.

Вычисления показывают, что $q_1\cdot q_2\cdot\ldots\cdot q_{15} \le 2\cdot 10^{18}$, а $q_1\cdot q_2\cdot\ldots\cdot q_{16} > 2\cdot 10^{18}$. Значит, число различных простых делителей в разложении n не превосходит 15. Поэтому

$$\frac{\sigma(n)}{n} \le \frac{q_1}{q_1-1}\dots\frac{q_{15}}{q_{15}-1} < 7.21.$$

Откуда, в частности,

$$\sigma(n) \le 7.21\cdot n \le 14.42\cdot 10^{18} < 2^{64}.$$

То есть для хранения величины s можно использовать беззнаковый 64-битный целый тип. При этом тесты подобраны так, что знакового для этого не хватает.

Кроме того, мы получаем оценку на число итераций цикла решения:

$$p^3 \le \frac{\sigma(d)+1}{d}N \le \left(\frac{\sigma(d)}{d}+1\right)N \le 8.21\cdot N.$$

Откуда

$$p \le \sqrt[3]{8.21\cdot N} \le \sqrt[3]{16.42}\cdot 10^6 \le 3\cdot 10^6.$$

То есть число итераций вполне допустимо при таких ограничениях. Сама же сложность алгоритма видимо равна $O(\sqrt[3]{N}\log\log N)$. Однако для получения этой оценки необходим более глубокий теоретико-числовой анализ оценки $\frac{\sigma(n)}{n} \le \frac{q_1}{q_1-1}\dots\frac{q_k}{q_k-1}$.

Задача I. Степени перестановок. Нам надо найти количество перестановок порядка N, каждая из которых является d-той степенью какой-то другой перестановки. Найдем сначала критерий того, что перестановка является d-той степенью.

Пусть перестановка p разложена на циклы. Проанализируем разложение на циклы перестановки p^d. Ясно, что p^d совпадает с объединением степеней c^d по всем циклам c перестановки p. Рассмотрим цикл c длины L из разложения p. Его можно представить как последовательность чисел $a_0, a_1, \dots, a_{L-1}$, причем перестановка p как отображение переводит a_i в a_{i+1} $(0 \le i < L-1)$, и a_{L-1} в a_0. Тогда ясно, что отображение c^d переводит a_i в $a_{(i+d) \bmod L}$. Легко видеть, что такое отображение распадается на $g = \gcd(d, L)$ циклов длины L/g.

Теперь сделаем обратный анализ. Пусть в p^d есть циклы длины l. Тогда для некоторых g и L должны выполняться соотношения: $L = lg$, $g = \gcd(d, L)$, что равносильно выполнению равенства $1 = \gcd(d/g, l)$. Пусть $p_1, p_2, \dots, p_s$ общие простые делители d и l, причем p_i входит в разложение d на простые множители с показателем α_i. Тогда ясно, что условие $1 = \gcd(d/g, l)$ равносильно тому, что g делится на $p_1^{\alpha_1}p_2^{\alpha_2}\dots p_s^{\alpha_s}$.

Обозначим это число как $h(d,l)$. Все циклы длины l разбиваются на группы, которые соответствуют разным g, и в каждой группе их количество кратно g. Поэтому общее число циклов длины l кратно $h(d,l)$.

Таким образом, мы получаем следующее необходимое условие того, что перестановка является d-той степенью: для каждого l количество циклов длины l кратно $h(d,l)$. Докажем, что оно является и достаточным. Пусть q – перестановка, и для нее выполнено это условие. Построим перестановку p, для которой $p^d = q$. Пусть $1 \le l \le N$ и k_l – количество циклов длины l в разложении q на циклы. По условию k_l делится на $g = h(d,l)$. Разобьем все циклы длины l на группы по g циклов любым способом. Пусть $c_1 = (a_{1,0}, a_{1,1}, \ldots, a_{1,l-1})$, $c_2 = (a_{2,0}, a_{2,1}, \ldots, a_{2,l-1})$, $\ldots$, $c_g = (a_{g,0}, a_{g,1}, \ldots, a_{g,l-1})$ – одна из таких групп. Рассмотрим цикл $c = (a_{1,0}, a_{2,0}, \ldots, a_{g,0}, a_{1,1}, a_{2,1}, \ldots, a_{g,1}, \ldots, a_{1,l-1}, a_{2,l-1}, \ldots, a_{g,l-1})$. Ясно, что в степени g он будет распадаться на циклы $c_1, c_2, \ldots, c_g$. По свойствам $h(d,l)$ выполнено равенство $\gcd(l, d/g) = 1$, и по построению $h(d,l)$ выполнено равенство $\gcd(g, d/g) = 1$. Значит, $\gcd(lg, d/g) = 1$. Поэтому из цикла c можно извлечь корень d/g-той степени, и он тоже будет циклом длины lg. Объединяя все такие циклы для всех групп и для всех l получим искомую перестановку p.

Теперь уже решить саму задачу не составляет труда. Пусть $f(n,l)$ – количество красивых перестановок длины n, в которых все циклы имеют длину $\le l$. Тогда ответ на задачу это $f(N,N)$. Мы хотим выразить $f(n,l)$ через предыдущие значения. Переберем количество циклов длины l. Так как оно кратно $h(d,l)$, то перебирать надо числа k вида $h(d,l)j$, $0 \le j \le \left\lfloor \frac{n}{h(d,l)l} \right\rfloor$ (последнее неравенство выполняется так как сумма длин всех циклов равна n). Для фиксированного k количество способов выбрать k циклов длины l равно

$$\frac{n!}{(n-kl)!k!l^k}$$

(Один из способов объяснения. Сначала выбираем упорядоченный набор из kl чисел из n, что можно сделать $A_n^{kl} = \frac{n!}{(n-kl)!}$ способами. Если мы выпишем выбранные числа подряд и возьмем в скобочки каждые l, то получим разложение на циклы, но так как порядок циклов не важен и с какого числа начинать запись цикла также не имеет значения, то надо поделить это число еще на $k!l^k$).

Если мы выбросим выбранные циклы из перестановки, то получим произвольную красивую перестановку порядка $n - kl$, в которой все циклы имеют длину $\le l-1$. Значит, перестановки, в которых ровно k циклов длины l дают общий вклад

$$f(n-kl, l-1)\frac{n!}{(n-kl)!k!l^k}.$$

В итоге, получаем формулу

$$f(n,l) = \sum_k f(n-kl, l-1)\frac{n!}{(n-kl)!k!l^k},$$

где сумма ведется по всем значениям k, которые делятся на $h(d,l)$ и удовлетворяют неравенству $0 \le k \le n/l$.

Так как ответ требуется вычислить по большому простому модулю, то деления следует проводить, используя обратные остатки, то есть рабочая формула имеет вид

$$f(n,l) = \sum_k f(n-kl, l-1)n!\mathrm{inv}((n-kl)!k!l^k),$$

где суммирование такое же, как и раньше, а $\mathrm{inv}(x)$ удовлетворяет свойству $x\mathrm{inv}(x) \equiv 1 \pmod P$, где $P = 1000000009$. $\mathrm{inv}(x)$ можно, например, найти по формуле $\mathrm{inv}(x) = x^{P-2} \bmod P$, которая справедлива в силу малой теоремы Ферма.

Заметим также, что мы можем сделать небольшое неасимптотическое ускорение. Перепишем формулу для $f(n,l)$ в виде

$$\frac{f(n,l)}{n!} = \sum_k \frac{f(n-kl, l-1)}{(n-kl)!}\mathrm{inv}(k!l^k).$$

Введем новую функцию $g(n,l) = f(n,l)/n!$ – долю красивых перестановок среди всех. Тогда ответ на задачу $g(N,N)N!$, а формула для $g(n,l)$ имеет вид

$$g(n,l) = \sum_k g(n-kl, l-1)\,\mathrm{inv}(k!l^k).$$

Значения $\mathrm{inv}(k!l^k)$ нужно предподсчитать в массив, так как каждый раз вызывать функцию вычисления обратного остатка не эффективно по времени. Кроме того, нужно предподсчитывать эти значения лишь для пар k, l, для которых $kl \le N$, то есть для $O(N \log N)$ пар. Легко организовать вычислительный процесс так, чтобы было выполнено $O(N(\log P + \log N))$ операций.

Количество итераций для вычисления $g(n, l)$ для всех l от 1 до n не превосходит суммы $n + n/2 + n/3 + \ldots + n/n = O(n \log n)$. Значит, общая сложность алгоритма равна $O(\sum_{n=1}^{N} n \log n) = O(N^2 \log N)$, что при ограничениях $N \le 2000$ вполне приемлемо.