Задача A. Загадочные числа

Идея. Идём слева направо и на каждом шаге выбираем минимальное число, которое (1) строго больше предыдущего и (2) подходит под текущий шаблон. Жадность корректна: чем меньше взяли сейчас, тем проще выполнить условие «строго больше» дальше.

Как найти минимальный подходящий кандидат быстро. Пусть длина шаблона m, тогда допустимый диапазон:

$$[10^{m-1}, 10^m - 1].$$

Стартуем с $x = \max(prev + 1, 10^{m-1})$ и проверяем соответствие шаблону по цифрам с конца. Если в позиции стоит `?` — без ограничений. Если стоит цифра и она не совпала, то вместо x++ делаем скачок по разряду:

пусть конфликт в разряде 10^k (младший $k = 0$), нужно цифру d. Фиксируем более старший префикс $P = \lfloor x/10^{k+1} \rfloor$ и берём число

$$y = P \cdot 10^{k+1} + d \cdot 10^k,$$

а младшие k разрядов ставим нулями (это минимизирует число при фиксированном конфликтном разряде). Если $y < x$, значит нужная цифра в этом префиксе «уже позади», тогда увеличиваем P на 1 (то есть прибавляем 10^{k+1}) и повторяем. Так мы попадаем в первое число $\geqslant x$, где конфликтный разряд может совпасть, и пропускаем диапазон заведомо неподходящих кандидатов.

Почему скачок ничего не пропускает. Пока не «перелистнём» конфликтный разряд на нужную цифру, совпадения быть не может, значит все пропущенные числа гарантированно плохие.

Задача B. Деревянное поле

Если $\min(n, m) = 1$, печатаем всё `.` — это путь, значит дерево, а стен нет.
Дальше считаем $n > 1$ и $m > 1$.

Конструкция. Нумеруем строки $i = 0..n-1$ и столбцы $j = 0..m-1$.
Задаём сдвиг

$$s = \begin{cases} 1, & n \bmod 6 \in \{1, 4\}, \\ 0, & \text{иначе}, \end{cases} \qquad r = i + s.$$

Клетка (i, j) равна `#` тогда и только тогда, когда выполняется одно из условий:

- $j = 0$ и $r \mod 6 = 4$;
- $j > 0$, $r \mod 6 \in \{0, 2\}$ и j нечётный;
- $j > 0$, $r \mod 6 \in \{3, 5\}$ и j чётный.

Во всех остальных случаях ставим `.`

Пример, $n = 8$, $m = 9$**:**

```
.#.#.#.#.
.........
.#.#.#.#.
..#.#.#.#
#........
..#.#.#.#
.#.#.#.#.
.........
```

Почему выполняются условия.

(1) Нет соседних стен. По горизонтали в строках с чередованием `#` стоят через одну клетку. По вертикали соседние типы строк никогда не используют одни и те же столбцы для `#`: типы $\{0, 2\}$ ставят `#` на нечётных $j > 0$, типы $\{3, 5\}$ — на чётных $j > 0$, а тип 4 имеет единственную стену в $j = 0$ (где у остальных типов по правилу первого столбца стен нет).

(2) Точки образуют дерево. Рассмотрим любые 6 подряд идущих строк по $r \bmod 6$. Внутри такого блока есть две «магистрали» из точек: строка типа 1 — сплошная по всем j, и строка типа 4 — сплошная по $j \geqslant 1$ (в $j = 0$ там специально стоит стена). Все остальные точки в строках типов $0, 2, 3, 5$ (в столбцах $j \geqslant 2$) не имеют горизонтальных связей и цепляются к одной из магистралей ровно одним вертикальным ребром, то есть это листья. Единственная связь между двумя магистралями проходит через область около столбцов 0–1; из-за стены в (тип $4, j = 0$) альтернативного обхода, который замкнул бы цикл, нет. Соседние 6-блоки по вертикали тоже склеиваются «узко» (через первый столбец), поэтому цикл при склейке не появляется. Итого: граф точек связен и без циклов, значит это дерево.

Задача C. Неизвестная перестановка

Идея. Вместо угадывания значений строим систему сравнений вида $p[u] < p[v]$. Если сравнения не противоречат (нет циклов), то любая топологическая нумерация даёт перестановку.

Ограничения из a. Поддерживаем представителей уровней $1, 2, \ldots$ (идея «слоёв» LIS). Для позиции i с $a[i] = k$:

- чтобы длина LIS, заканчивающейся в i, была хотя бы k, нужно связать i с представителем уровня $k - 1$ сравнением «тот меньше i»;
- чтобы i не мог «вылезти» выше k за счёт уже построенной структуры, мы обновляем представителя уровня k так, чтобы он оставался «самым выгодным хвостом», и добавляем сравнение, фиксирующее правильный порядок между новым и старым представителем (для каждых двух индексов $i < j$, если $a_i = a_j$, то $p_i > p_j$).

Если внезапно требуется уровень, который не может существовать на текущем префиксе, сразу `IMPOSSIBLE`.

Ограничения из b. Те же действия, но идём справа налево: так «убывающие вперёд» ограничения превращаются в аналогичную работу со слоями на суффиксе.

Финиш. Строим ориентированный граф сравнений. Если в нём цикл — `IMPOSSIBLE`. Иначе берём топологический порядок вершин и присваиваем значения $1..n$ в порядке топосорта, чтобы каждое ребро $u \to v$ означало $p[u] < p[v]$.

Задача D. Еще раз дерево

Нумеруем операции синхронизации по времени: каждое ребро e встречается в $\mathbf{S}$ ровно один раз, значит у него появляется метка $t(e) \in \{1, 2, \ldots, n - 1\}$ — момент синхронизации (чем раньше синхронизировали, тем меньше метка).

Ключевой критерий (почему вообще всё сводится к меткам). Фрагмент может «пересечь» ребро только в момент его синхронизации: до этого ребра как канала передачи нет, а после — событие уже прошло и заново по нему ничего «не проталкивается» в прошлое времени.

Пусть хотим понять, окажется ли фрагмент d в вершине u. Рассмотрим единственный простой путь $d \to u$ в дереве:

$$d = v_0 - v_1 - \cdots - v_k = u, \qquad e_i = (v_{i-1}, v_i).$$

Факт. Фрагмент d окажется в u <u>тогда и только тогда</u>, когда

$$t(e_1) < t(e_2) < \cdots < t(e_k).$$

Почему.

- ($\Rightarrow$) если фрагмент реально дошёл, то он пересёк e_1 в момент $t(e_1)$, потом должен был пересечь e_2, но сделать это можно только в момент $t(e_2)$, причём фрагмент обязан уже находиться в v_1 к этому моменту, значит $t(e_2) > t(e_1)$, и так далее.
- ($\Leftarrow$) если времена на пути строго возрастают, то в момент $t(e_1)$ фрагмент перейдёт из v_0 в v_1, затем (поскольку $t(e_2) > t(e_1)$) дождётся синхронизации e_2 и перейдёт дальше, и так по цепочке до u.

После этого задача превращается в **онлайновые запросы на дереве с временными метками на рёбрах**:

- $\mathbf{S}(u, v)$: присвоить ребру (u, v) новую метку t (текущее время);
- $\mathbf{Q}(u, d)$: проверить существование пути $d \to u$ со строго возрастающими метками;
- $\mathbf{C}(d)$: посчитать число вершин u, для которых путь $d \to u$ строго возрастает.

Как быстро отвечать на $\mathbf{Q}(u, d)$ (проверка монотонности по пути).
Удобно зафиксировать корень дерева (например, 1). Тогда каждое ребро можно ассоциировать с дочерней вершиной: для вершины $x \neq 1$ пусть $w(x)$ — метка ребра $(x, \mathrm{par}(x))$ (если ребро ещё не синхронизировано, считаем $w(x) = +\infty$, то есть пройти нельзя).

Запрос $\mathbf{Q}(u, d)$ проверяет, что на пути $d \to u$ последовательность $w(\cdot)$ вдоль рёбер строго возрастает в направлении движения. Пусть $L = \mathrm{lca}(d, u)$. Тогда путь распадается на две части:

$$d \to L \quad \text{(движение вверх)}, \qquad L \to u \quad \text{(движение вниз)}.$$

Условие «строго возрастает на всём пути» эквивалентно трём проверкам:

1. на отрезке $d \to L$ метки строго возрастают при движении вверх;
2. на отрезке $L \to u$ метки строго возрастают при движении вниз;
3. последняя метка на $d \to L$ меньше первой метки на $L \to u$ (чтобы склейка не ломала строгость).

Чтобы проверять это онлайново, удобно разложить любой путь на $O(\log n)$ отрезков по цепям (например, через разложение на тяжёлые пути). На каждом отрезке (в фиксированном направлении) достаточно уметь хранить агрегат:

$$(\text{first},\ \text{last},\ \text{ok}),$$

где `ok` означает «внутри отрезка метки строго возрастают в этом направлении», а `first/last` — первая/последняя метка на отрезке. Два соседних агрегата склеиваются за $O(1)$:

$$\mathtt{ok}_{12} = \mathtt{ok}_1 \wedge \mathtt{ok}_2 \wedge (\mathtt{last}_1 < \mathtt{first}_2),$$

и новые `first/last` берутся очевидно. Тогда проверка пути — это сборка агрегатов в правильном порядке на $O(\log n)$ кусках, что даёт $O(\log^2 n)$. Обновление $\mathbf{S}$ — это присвоение одной метки $w(x)$, то есть точечное обновление структуры за $O(\log n)$ на одну цепь.

Почему запрос $\mathbf{C}(d)$ сложный. Нужно посчитать число u, для которых выполнено «$d \to u$ строго возрастает», нельзя просто перебрать все u и вызвать $\mathbf{Q}$.

Ключевая идея: считать ответы $\mathbf{C}$ через **centroid decomposition**.

Разложение по центроидам: как превращаем подсчёт в «счёт по порогу».
Возьмём центроидное разложение исходного дерева. Для каждой вершины x есть цепочка центроидных предков

$$x = c_0,\ c_1, \ldots, c_k \quad (k = O(\log n)).$$

Классический трюк: чтобы посчитать количество вершин, удовлетворяющих свойству относительно фиксированного источника d, можно суммировать вклад по этим c_i, а чтобы не пересчитать вершины несколько раз, вычитать вклад «детского компонента», через который d входит в центроид (стандартное включение–исключение в центроидном дереве).

Нам нужно понять, когда вершина u достижима из d через некоторый центроид c. Путь $d \to u$ проходит через c тогда и только тогда, когда c лежит на этом пути (в исходном дереве). Если это так, то путь можно склеить как

$$d \to c \to u.$$

Для строгого возрастания меток достаточно:

- чтобы кусок $d \to c$ был строго возрастающим; обозначим его последнюю метку как $L(d, c)$ (если кусок невозможен, считаем $L(d, c) = +\infty$);
- чтобы кусок $c \to u$ был строго возрастающим; в этом случае все его метки строго возрастают, а значит задаются первой меткой $F(c, u)$ (метка первого ребра при выходе из c в сторону u);
- и чтобы при склейке выполнялось $L(d, c) < F(c, u)$.

То есть для фиксированного d и c задача сводится к:

$$\text{посчитать } u \text{ такие, что } (c \to u \text{ возрастает}) \ \wedge \ F(c, u) > L(d, c).$$

Это уже типичный вид «посчитать сколько значений F больше заданного порога».

Что храним в каждом центроиде.

Для каждого центроида c поддерживаем структуру (мультимножество по времени) по вершинам u, для которых путь $c \to u$ является строго возрастающим:

- ключ элемента — $F(c, u)$ (первая метка на пути $c \to u$);
- запрос — сколько ключей $> X$ (где $X = L(d, c)$).

Чтобы делать это быстро, времена t лежат в $[1..n-1]$, их удобно сжать и хранить частоты в дереве Фенвика/Дереве Отрезков, тогда запрос «сколько $> X$» — $O(\log n)$.

Чтобы избежать двойного счёта в сумме по центроидным предкам c источника d, дополнительно для каждого c держим такие же структуры по каждому его ребёнку в центроидном дереве (или по компонентам).

Что происходит при операции $\mathbf{S}(u, v)$.

При синхронизации ребра (u, v) ему присваивается новое время t. Это может сделать некоторые пути строго возрастающими (раньше ребро было «недоступно», теперь оно стало доступно с конкретной меткой). Важно: метки никогда не меняются, только появляются, поэтому любое свойство «путь стал возможен» происходит монотонно: путь может стать допустимым, но не может перестать им быть.

Поэтому обновления делаются так:

- обновляем значение $w(\cdot)$ для одной вершины (ребро к родителю) — это точечное обновление для структур, отвечающих за $\mathbf{Q}$;
- для структур в центроидах: как только для пары (c, u) становится истинным «путь $c \to u$ строго возрастает», мы добавляем ключ $F(c, u)$ в структуру центроида c (и в структуру соответствующей компоненты для включения–исключения).

Так как у каждой вершины u всего $O(\log n)$ центроидных предков, суммарное число вставок — $O(n \log n)$, а каждая вставка/запрос по времени — $O(\log n)$.

Как считается $\mathbf{C}(d)$.

Идём по центроидным предкам c вершины d. Для каждого c:

- проверяем, достижим ли c из d (то есть кусок $d \to c$ строго возрастает); если нет — этот c не даёт вклада;
- иначе считаем $X = L(d, c)$ и добавляем

$$\#\{u : \ F(c, u) > X\}$$

из глобальной структуры c;

Итого получаем количество вершин, достижимых из d по критерию строгого возрастания меток.

Итоговая оценка.

- **Q**: разложение пути на $O(\log n)$ кусочков $\Rightarrow$ около $O(\log^2 n)$;
- **C**: $O(\log n)$ центроидов, в каждом запрос по времени $O(\log n) \Rightarrow O(\log^2 n)$;
- **S**: одно точечное обновление для **Q**-структур и $O(\log n)$ вставок/обновлений по центроидам, каждая за $O(\log n) \Rightarrow$ тоже $O(\log^2 n)$.