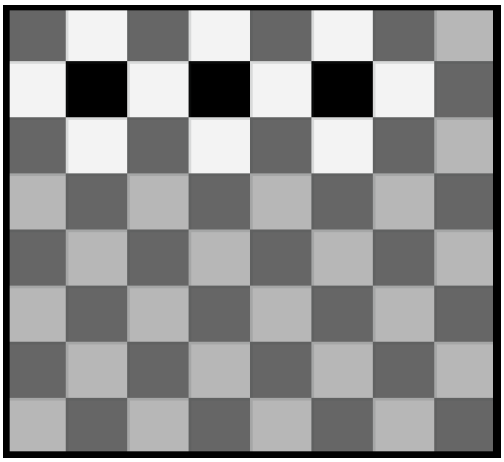
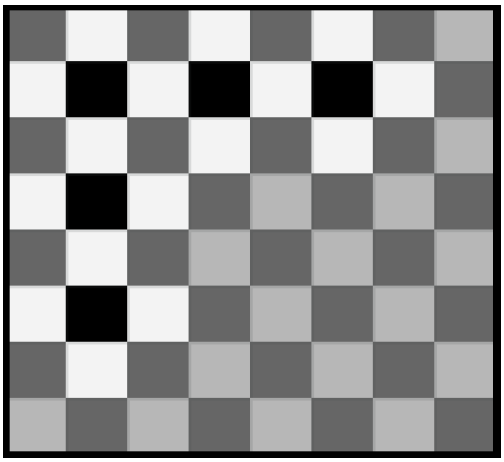
# Разбор задачи «Designing a New Logo»

В первой подзадаче можно было начать с левой верхней клетки и жадно выбрать 2 соседние по горизонтали чёрные и белые клетки, связные с текущей областью.

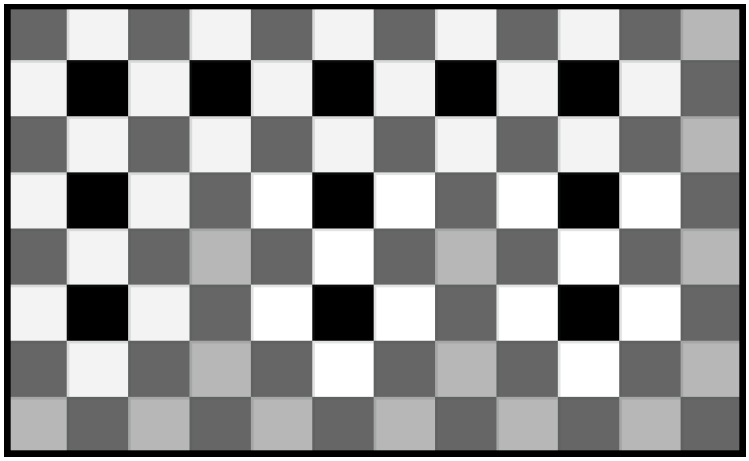
Во второй подзадаче можно было выбрать клетки с координатами $(2,2),(2,3),(2,4),\dots(2,2b)$ (получив $b$ чёрных и $b-1$ белых), а потом выбрать белые клетки, соседние с выбранными чёрными. Всего дополнительно мы сможем выбрать до $2b+2$ белых клеток, поэтому гарантированно получим ответ.



В третьей подзадаче можно было выбрать сначала клетки вида $(2,j)$ $(2 \leqslant j \leqslant 4m-2)$, а потом клетки вида $(i,2)$ $(3 \leqslant i \leqslant 4n-2)$, а потом выбрать белые клетки, соседние с выбранными чёрными. Итого можем получить до $b=(2m-1)+(2n-2)$ чёрных клеток, между которыми $b-1$ белых, и мы сможем выбрать до $2b+2$ дополнительных белых клеток.



В четвёртой подзадаче можно было выбрать сначала клетки вида $(2,j)$ $(2 \leqslant j \leqslant 4m-2)$, а потом клетки вида $(i,2+4j)$ $(3 \leqslant i \leqslant 4n-2, 2 \leqslant 2+4j \leqslant 4m-2)$, а потом выбрать белые клетки, соседние с выбранными чёрными. Итого можем получить до $b=(2m-1)+(2n-2)\cdot m = 2nm-1 > nm$ чёрных клеток, между которыми $b-1$ белых, и мы сможем выбрать до $2b+2$ дополнительных белых клеток.



# Разбор задачи «Even Tree»

Для начала заметим, что в задаче просят найти остовное дерево, в котором будет четное количество ребер нечетного веса. Рассмотрим компоненты связности только по четным ребрам. Внутри каждой компоненты мы точно можем выбрать остовное дерево из всех четных ребер, а также остовное дерево ровно с одним нечетным ребром, если оно есть внутри компоненты. Таким образом, получаем несколько случаев:

1. У нас нечетное количество компонент связности по четным ребрам — тогда нам нужно построить внутри каждой компоненты остов из всех четных ребер и соединить все компоненты;
2. У нас четное количество компонент и нет нечетных ребер внутри компоненты (то есть между двумя различными вершинами из одной компоненты) — тогда у нас не получится построить остов четного веса;
3. У нас четное количество компонент и есть нечетное ребро внутри компоненты — тогда мы должны взять это ребро, а остальные ребра взять так же, как и в первом случае (то есть у нас будет замена одного четного ребра на нечетное).

Это можно реализовать, например, следующим образом: построим дерево обхода в порядке dfs'а. Если это дерево нам уже подоходит (то есть суммарный вес ребер четный), то берем его как ответ. Иначе нам нужно найти обратное ребро, которое можно будет заменить на ребро другой четности. Если у нас не получилось найти такую пару ребер, значит, ответа не существует.

# Разбор задачи «Primle»

Разберем все подзадачи по порядку.

В первой и второй подзадаче можно спрашивать случайные запросы 99 раз, а дальше пытаться восстановить ответ. Можно показать, что с большой вероятностью мы получим `+` в каждой из пяти позиций хотя бы один раз, поэтому мы сможем восстановить загаданное число. В сложной версии нужно уметь генерироавть случайное простое число.

В третьей и четвертой подзадачах можно придумать такой набор из 9 чисел, что на каждой позиции побывает 9 из 10 возможных цифр. В простом режиме это набор `11111`, `22222`, ..., `99999`. В сложном режиме нужно подобрать девять простых чисел с такими свойствами. По этой информации тоже однозначно восстанавливается каждый разряд.

В пятой и шестой подгруппе можно соптимизироавть предыдущее решение, первыми двумя запросами спросив `65423` и `91807`. После этих двух запросов мы знаем множество цифр, которое присутствует в числе. Затем, так же, как в прошлой подзадаче подобрать 4 запроса так, чтобы на каждой позиции однозначно определить число.

Наконец, последние две подгруппы можно решить несколькими способами, опишем один из них. На каждом шаге у нас есть множество простых чисел, которые подходят под всю известную информацию к текущему моменту. Каждый новый запрос разбивает это множество чисел на $3^5$ подмножеств, из которых мы оставим одно. Когда в допустимом множестве чисел осталось одно число,

выдаем его, как ответ. Поскольку мы хотим гарантированно угадывать каждое загаданное простое число за $L$ запросов, то логично максимизировать "худший"случай и разбивать таким образом, чтобы максимальный размер из 243 групп был минимальным. Если следовать такой стратегии, то почти все простые отгадаются за $\leqslant 4$ запроса, и для всего около 50 простых нужно будет 5 запросов.

Бонус: придумайте стратегию, которая отгадывает любое простое гарантированно за 4 запроса.

# Разбор задачи «Add and Multiply»

Для начала рассмотрим случай, где для всех $i$ $a_i \leqslant b_i$ или для всех $i$ $a_i \geqslant b_i$. Это значит что если произведения не равны, то никаким образом их сравнять невозможно. Если же произведения равны, то можно вывести все $c_i = 0$.

Теперь докажем, что если существуют такие $i, j$, что $a_i < b_i$ и $a_j > b_j$, то подходящее решение гарантированно существует.

Для начала решим задачу для $n = 2$. Не умаляя общности, скажем что $a_1 < b_1, a_2 > b_2$. Тогда ответом будет являться такие $c_1, c_2 \geqslant 0$, что $(a_1+c_1)(a_2+c_2) = (b_1+c_1)(b_2+c_2) \Rightarrow (a_1+c_1)(a_2+c_2)-(b_1+c_1)(b_2+c_2) = 0$. Раскроем скобки, и получим что $a_1a_2-b_1b_2+c_1(a_2-b_2)+c_2(a_1-b_1) = 0$ Получается, что $c_1(a_2-b_2)-c_2(b_1-a_1) = a_1a_2-b_1b_2$. Докажем, что у этого уравнения всегда существует целое неотрицательное решение.

Найдём решение с помощью расширенного алгоритма Евклида. Для того чтобы он работал, необходимо чтобы $a_1a_2 - b_1b_2$ делилось на $d = \gcd(a_2 - b_2, b_1 - a_1)$. Это верно, потому что $a_2 = b_2 \pmod d, b_1 = a_1 \pmod d \Rightarrow a_1a_2 - b_1b_2 = b_1b_2 - b_1b_2 = 0 \pmod d$. При этом $a_2 - b_2 > 0, b_1 - a_1 > 0$, поэтому решив $(a_2 - b_2)x - (b_1 - a_1)y = 0$ можно сделать из произвольного решение целое неотрицательное.

Теперь давайте решим задачу в общем случае. Для этого научимся объединять множество столбцов с $a_i > b_i$. Рассмотрим такой пример:

| | | |
|---|---|---|
| a | 9 | 10 |
| b | 5 | 3 |

Заметим, что если добавить ко второму элементу 6, пары уравняются:

| | | |
|---|---|---|
| a | 9 | 16 |
| b | 5 | 9 |

Теперь достаточно будет найти решение для $a = 16, b = 5$, а потом прибавить одно и то же значение ко всем столбцам.

В общем случае для всех столбцов с $a_i > b_i$ необходимо сначала отсортировать их по убыванию $b_i$, а потом по порядку выравнивать $a_i$ и $b_{i+1}$. Тогда на каждом шаге для выравнивания мы добавим целое неотрицательное число к $c_i$, так как $b_{i+1} \leqslant b_i < a_i$. В итоге мы можем за $O(n)$ приравнять все столбцы с $a_i > b_i$, а потом приравнять все столбцы с $a_i < b_i$ (это делается аналогично с точность до замены $a_i$ и $b_i$). Получим две пары чисел, для которых мы уже умеем решать задачу.

# Разбор задачи «Draft Laws»

В задаче требуется найти количество правильных раскрасок вершин дерева в $k$ цветов (по простому модулю) с учётом того, что некоторые вершины уже покрашены.

*Подзадачу 1* можно было решить полным перебором всех $k^n$ раскрасок.

*Подзадача 2* основана на том факте, что существует ровно два способа раскрасить дерево в два цвета: при фиксированном цвете у вершины 1 цвета всех остальных вершин однозначно восстанавливаются. Тогда можно перебрать обе раскраски и для каждой проверить, подходит ли она.

В *подзадаче 3* нет заранее покрашенных вершин. Легко убедиться в том, что ответ равен $k(k-1)^{n-1}$: для вершины 1 мы можем выбрать любой из $k$ цветов, далее, для всех соседей вершины 1 мы можем выбрать любой из $k-1$ оставшихся цветов; для каждой из вершин на расстоянии 2 от первой вершины снова есть $k-1$ вариантов выбора, и так далее.

Подзадачи, начиная с 4, можно решить методом динамического программирования. Подвесим дерево за вершину 1. Пусть $\mathrm{dp}_{v,c}$ — число способов раскрасить поддерево вершины $v$ (с учётом всех требований) так, что сама вершина $v$ покрашена в цвет $c$. Тогда, значение $\mathrm{dp}_{v,c}$ можно пересчитать, зная значения динамики для сыновей. В случае, когда $a_v = 0$ имеем:

$$\mathrm{dp}_{v,c} = \prod_{u\ —\ \text{сын}\ v} \sum_{d \neq c} \mathrm{dp}_{u,d}$$

поскольку для каждого сына $u$ можно независимо выбрать любой цвет $d \neq c$. В случае $a_v \neq 0$ пересчёт динамики абсолютно такой же за тем исключением, что $\mathrm{dp}_{v,c} = 0$ при $c \neq a_v$.

Наивный пересчёт этой динамики даёт решение за $O(nk^2)$ (для каждого из $n-1$ рёбер $(vu)$ мы перебираем $O(k^2)$ пар цветов $(c,d)$), решающее *подзадачу 4*.

Можно легко оптимизировать динамику следующим образом: при пересчёте $\mathrm{dp}_{v,c}$ и при фиксированном $u$, вместо того, чтобы наивно суммировать по всем $d \neq c$, давайте для всех вершин также считать $\mathrm{sum}_v = \sum_{c=1}^{k} \mathrm{dp}_{v,c}$, и тогда можно сразу будет взять $\mathrm{sum}_u - \mathrm{dp}_{u,c}$. Получаем решение за $O(nk)$, которое решает *подзадачу 5*.

Для решения остальных подзадач нужно избавиться от параметра $k$. Среди всех $k$ цветов лишь не более $n$ цветов встречаются на дереве (на уже покрашенных вершинах). Пусть $v$ — любая вершина. Заметим, что если $c_1$ и $c_2$ — два цвета, которые не встречаются в поддереве $v$, то верно равенство:

$$\mathrm{dp}_{v,c_1} = \mathrm{dp}_{v,c_2}$$

Действительно, можно установить взаимно однозначное соответствие между раскрасками поддерева $v$, где цвет $v$ равен $c_1$, и раскрасками, где цвет $v$ равен $c_2$: достаточно просто поменять местами цвета $c_1$ и $c_2$ (все вершины, покрашенные в $c_1$, перекрасить в $c_2$, и наоборот). От этого никакое требование не нарушится, и поэтому этих раскрасок поровну. Отсюда вытекает **ключевая идея**: для всех цветов, которые не встречаются в поддереве $v$ мы будем хранить одно общее значение динамики, которое обозначим $\mathrm{other}_v$.

Для решения *подзадачи 6* достаточно избавиться от цветов, отсутствующих на всём дереве. Возьмем все цвета, встречающиеся на дереве, и применим «сжатие координат»: номерам цветов, встречающихся на дереве, поставим в соответствие последовательные номера $0, 1, \ldots, D-1$, где $D$ — количество различных цветов на дереве. Затем посчитаем аналогичную динамику $\mathrm{dp}_{v,c}$, только уже при $0 \leqslant c < D$, и одновременно пересчитывая и учитывая $\mathrm{other}_v$. Таким образом, при $a_v \neq 0$, пересчёт будет проводиться по формуле:

$$\mathrm{dp}_{v,c} = \prod_{u\ —\ \text{сын}\ v} \left(\mathrm{sum}_u + (k-D)\cdot \mathrm{other}_u - \mathrm{dp}_{u,c}\right)$$

Это решение работает за $O(nD)$ (где $D$ — количество различных цветов на дереве), в частности время работы можно оценить как $O(n \cdot \min(n,k))$, или $O(n^2)$.

В *подзадаче 7* дерево является путём (бамбуком). Разобьём его на части, ограниченные покрашенными вершинами. Заранее посчитаем следующие величины: $A_m$ — количество раскрасок пути из $m$ вершин, где цвета крайних вершин фиксированы и совпадают; и $B_m$ — то же количество, но если цвета крайних вершин фиксированы и не совпадают. Их можно посчитать следующими рекуррентными формулами:

$$A_m = (k-1)B_{m-1}, \quad A_1 = 1$$
$$B_m = A_{m-1} + (k-2)B_{m-1}, \quad B_1 = 0$$

Для каждой из частей можно независимо выбрать $A_m$ или $B_m$ раскрасок внутренних вершин (в зависимости от того, одинаковый или разный цвет у крайних вершин), поэтому достаточно взять произведение этих количеств по всем частям.

Теперь перейдём к *полному решению*. Точно так же мы будем считать динамику по поддеревьям. Для вершины $v$ мы будем хранить значения динамики $\mathrm{dp}_{v,c}$ (для цветов $c$, встречающихся в поддереве $v$) в ассоциативном массиве $\mathrm{H}_v$ (например, std::map в C++), в котором ключами будут цвета

$c$, а значениями — соответствующие значения $\mathrm{dp}_{v,c}$. Также мы будем хранить $\mathrm{sum}_v$ — сумму всех значений в $\mathrm{H}_v$, $\mathrm{cnt}_v$ — количество значений в $\mathrm{H}_v$ и $\mathrm{other}_v$ — значения динамики для всех остальных цветов.

Воспользуемся методом «слияния меньшего к большему». Для каждой вершины $v$ выделим её сына $b$, у которого наибольший размер $\mathrm{H}_b$. Сначала мы за $O(1)$ положим $\mathrm{H}_v := \mathrm{H}_b$ (с некоторыми уточнениями, чтобы значения были корректными), а затем для всех остальных сыновей $u \neq b$ мы будем перекладывать все элементы из $\mathrm{H}_u$ в $\mathrm{H}_v$ наивно, проходясь по всему контейнеру $\mathrm{H}_u$. Заметим, что такой перебор цветов в остальных поддеревьях суммарно займет $O(n \log n)$ перекладываний. Мы всегда перекладываем элемент в контейнер с большим числом элементов, поэтому размер контейнера, содержащего цвет $c$, за одно перекладывание увеличивается хотя бы в два раза. Из этого следует, что цвет $c$ суммарно будет переложен не более $O(\log n)$ раз.

Сначала обсудим, как нам из значений, лежащих в $\mathrm{H}_b$, получить изначальные значения для вершины $v$. Для любого цвета $c$ из $\mathrm{H}_b$ мы хотим получить:

$$\mathrm{dp}_{v,c} := \mathrm{sum}_b + (k - \mathrm{cnt}_b) \cdot \mathrm{other}_b - \mathrm{dp}_{b,c}$$

Иными словами, если обозначить $T = \mathrm{sum}_b + (k - \mathrm{cnt}_b) \cdot \mathrm{other}_b$, то нужно сначала выполнить замену $\mathrm{H}_v := \mathrm{H}_b$, а затем для всех значений $x$ из $\mathrm{H}_v$ нужно применить линейную функцию:

$$x \mapsto -x + T$$

Подобные массовые операции можно выполнять лениво: вместо того, чтобы применять её ко всем значениям, вручную перебирая $\mathrm{H}_v$, мы просто сохраним эту линейную функцию рядом, обозначив её $f_v(x)$. Чтобы получить актуальное значение $\mathrm{dp}_{v,c}$, достаточно лишь будет к значению, которое фактически лежит в $\mathrm{H}_v$ применить функцию $f_v(x)$. Изначально $f_v(x)$ равна тождественной функции $x \mapsto x$. Когда нам приходит массовая операция вида «ко всем значениям в $\mathrm{H}_v$ применить линейную функцию $g(x) = px + q$», мы обновим коэффициенты функции $f_v(x)$ и значения $\mathrm{sum}_v$, $\mathrm{other}_v$ следующим образом (до этого было $f_v(x) = ax + b$):

$$\begin{aligned} f_v(x) &:= g(f_v(x)) = (pa)x + (pb + q) \\ \mathrm{sum}_v &:= p \cdot \mathrm{sum}_v + q \cdot \mathrm{cnt}_v \\ \mathrm{other}_v &:= p \cdot \mathrm{other}_v + q \end{aligned}$$

Таким образом, обработка сына $b$ сведена к одной массовой операции, которую мы выполним лениво за $O(1)$.

Теперь обработаем всех сыновей $u \neq b$. Если цвет $c$ не лежит в $\mathrm{H}_u$, то для него пересчёт выполняется следующим образом:

$$\mathrm{dp}_{v,c} := \mathrm{dp}_{v,c} \cdot (\mathrm{sum}_u + (k - \mathrm{cnt}_u - 1) \cdot \mathrm{other}_u)$$

Или, если обозначить $Q = \mathrm{sum}_u + (k - \mathrm{cnt}_u - 1) \cdot \mathrm{other}_u$:

$$\mathrm{dp}_{v,c} := \mathrm{dp}_{v,c} \cdot Q$$

А если цвет $c$ лежит в $\mathrm{H}_u$, то:

$$\mathrm{dp}_{v,c} := \mathrm{dp}_{v,c} \cdot (\mathrm{sum}_u + (k - \mathrm{cnt}_u) \cdot \mathrm{other}_u - \mathrm{dp}_{u,c})$$

Поступим следующим образом: сначала ко всем значениям в $\mathrm{H}_v$ применим линейную функцию $x \mapsto Qx$. После этого, вручную переберём все цвета $c$ из поддерева $u$, и для каждого из них выполним преобразование

$$\mathrm{dp}_{v,c} := \mathrm{dp}_{v,c} \cdot Q^{-1} \cdot (\mathrm{sum}_u + (k - \mathrm{cnt}_u) \cdot \mathrm{other}_u - \mathrm{dp}_{u,c})$$

где $Q^{-1}$ — обратный к $Q$ элемент по модулю $M = 10^9 + 7$. Проблема возникает, когда $Q \equiv 0 \pmod{M}$. Но в этом случае можно поступить иначе: давайте сначала для всех цветов $c$ из $\mathrm{H}_u$ заранее

посчитаем новое значение $\mathrm{dp}_{v,c}$ и сохраним его. Когда происходит массовая операция «умножить все значения из $\mathrm{H}_v$ на 0», все значения в $\mathrm{H}_v$ станут равными нулю. Поэтому, можно вообще полностью очистить $\mathrm{H}_v$, а также присвоить $\mathrm{sum}_v = \mathrm{other}_v = 0$ и положить $f_v(x)$ равной тождественной функции $x \mapsto x$. После этого, можно снова пройтись по цветам $c$ из $\mathrm{H}_u$, и присвоить ранее сохранённые значения.

В итоге, мы получаем решение за $O(n \log^2 n)$. Отметим, что можно добиться оценки $O(n \log n)$, используя std::unordered_map вместо std::map, но на практике это работает медленнее.