Задача 5. Обобщённые шахматы

Краткое условие задачи Дана доска $n \times n$ ($1 \leqslant n \leqslant 400$), клетка (i, j) покрашена в цвет a_{ij}. Для каждой пары r, c ($1 \leqslant r, c \leqslant n$) нужно найти минимальное количество клеток для перекраски b_{rc}, необходимое, чтобы прямоугольник из первых r строк и первых c столбцов стал *правильно раскрашенным*:

- используется не более двух разных цветов;
- соседние по стороне клетки имеют разные цвета.

Краткая идея решения

Для фиксированного прямоугольника (r, c) нужно выбрать два цвета c_0, c_1 (для клеток с чётной и нечётной суммой координат). Для каждой чётности нам нужны два самых частых цвета. В полном решении используется сжатие цветов и поддержка двух максимумов за $O(1)$, что даёт $O(n^3)$ времени на полное решение.

Подробный разбор:

Подзадача 1. ($n \leqslant 50$)

Вместо того, чтобы считать количество клеток, которые нужно перекрасить — будем наоборот считать количество клеток, которые не поменяют цвет. Количество таких клеток мы хотим максимизировать. У нас есть два типа клеток (i, j) — у которых $(i + j) \bmod 2 = 0$ и $(i + j) \bmod 2 = 1$, и они должны быть разных цветов. Зафиксируем подпрямоугольник (r, c), для которого мы решаем задачу. Нам нужно зафиксировать два цвета col_1 и col_2, чтобы максимизировать количество клеток с $(i + j) \bmod 2 = 0$ цвета col_1 + количество клеток с $(i + j) \bmod 2 = 1$ цвета col_2.

Сделаем структуру данных, в которую мы будем уметь добавлять пару (col, cnt) и поддерживать два максимальных cnt у разных цветов col. Переберем все клетки подпрямоугольника (r, c) и посчитаем для каждого цвета и типа клетки (остатка по модулю два) количество вхождений. После этого посчитаем два максимума для клеток типа 1 и типа 2 (с помощью структуры, которую мы определили выше). Теперь, если самый частый цвет среди клеток типа 1 и клеток типа 2 различаются, то нам выгодно оставить их. Иначе нам нужно рассмотреть два варианта: второй по частоте цвет среди клеток типа 1 + самый частый цвет среди клеток типа 2, и самый частый цвет среди клеток типа 1 + второй по частоте цвет клеток типа 2.

Возьмем максимум среди этих возможных вариантов, и ответом для подпрямоугольника (r, c) будет rc − количество клеток, которые сохранили цвет. Асимптотика $O(n^4 \log n)$, где log для того, чтобы считать для каждого цвета количество его вхождений.

Подзадача 2. ($n \leqslant 200$)

Эта подзадача близка к полному решению, но в ней можно реализовать структуру, которая поддерживает два максимума за log, либо реализовать структуры данных не самым оптимальным образом.

Зафиксируем r и найдем ответ для всех подпрямоугольников (r, c). Будем перебирать c по возрастанию и поддерживать для каждого цвета и типа клетки, количество вхождений этого цвета в текущий подпрямоугольник (r, c). Также, будем для обоих типов поддерживать два самых частых цвета, как в подзадаче 1. При переходе от c к $c + 1$ честно добавим r клеток, которые добавляются при расширении прямоугольника.

Таким образом, получаем решение за $O(n^3 \log n)$, где log от допустимой неэффективной реализации структуры с двумя максимума на базе map или set.

Подзадача 3. (Только два различных цвета)

В этой подзадаче всего два цвета, поэтому для каждого подпрямоугольника у нас всего две возможные цветовые конфигурации и мы можем явно посчитать ответ для каждой из них за $O(n^2)$.

Подзадача 4 (не более 10 разных цветов)

В этой подзадаче мы можем перебрать пару цветов, в которые будет раскрашена доска и для каждого подпрямоугольника проверить конфигурацию за $O(n^2)$. Таким образом, время работы $O(10^2 \times n^2)$.

Подзадача 5 (не более 100 разных цветов)

Эту группу тестов могли проходить решения с недостаточно эффективной реализацией следующих подгрупп.

Подзадача 6 (не более 10000 разных цветов)

В этой подзадаче ожидается полное решение, за исключение того, что чтобы поддерживать количество вхождений какого-то цвета, достаточно создать массив на 10000 элементов, вместо поддержки счетчиков за логарифм.

Полное решение

Сначала сожмем координаты, то есть сделаем так, чтобы все числа оказались меньше, чем n^2. Сжатие координат можно реализовать с помощью хеш-таблицы или с помощью сортироваки и бинарного поиска. Этой займет время $O(n^2)$ или $O(n^2 \log n)$. После этого реализуем решение из подзадачи 2, в котором мы за $O(1)$ поддерживаем количество вхождений какого-то цвета и за $O(1)$ поддерживаем два максимума. Получаем решение за $O(n^3)$, которое без проблем сдается.

Задача 6. Ночь, улица, фонарь, аптека

Подгруппа 1: $t = 1$, $n \leqslant 10$

Переберём все подмножества фонарей. Осталось проверить, что подмножество фонарей является экономным. Для этого можно перебрать для каждого фонаря, смотрит он вправо или влево, и явно проверить, выполняется ли условие их экономности. Данный перебор работает за $O(3^n)$, так как каждому перебранному варианту может соответствовать троичная маска: 0 - мы не взяли фонарь в экономное множество, 1 — мы взяли фонарь, и он направлен влево, 2 - мы взяли фонарь, и он направлен вправо.

Подгруппа 2: $t = 1$, $x_i - s_i \neq x_j$ и $x_i + s_i \neq x_j - s_j$

Описанное условие означает, что все фонари на отрезке нужно направить вправо. Будем считать $dp_{i,x}$ — количеством подмножеств фонарей, если мы рассматриваем фонари с номерами $\geqslant i$, и подмножество можно направить вправо так, чтобы самый левый освещенный отрезок начинался в позиции x.

При пересчёте $dp_{i,x} = dp_{i+1,x}$, кроме ситуации, когда мы используем i-й фонарь. В таком случае получается обновление $dp_{i,x_i} = 1 + dp_{i+1,x_i+s_i}$.

Получаем решение за $O(n + X)$, где X — максимальное ограничение на координаты.

Подгруппа 3: $t = 1$, $s_i \neq s_j$

Рассмотрим, как устроен произвольный ответ. Сначала сколько-то первых фонарей у нас направлены вправо. Далее, если дальше фонарь был направлен влево, то теперь следующие фонари должны освещать его правый конец, тогда все оставшиеся фонари светят влево. Таким образом, у нас сначала сколько-то фонарей направлены вправо, а потом сколько-то влево.

Заметим, что в этой подгруппе каждое подмножество можно определить единственным способом (кроме случая, если оно состоит ровно из одного фонаря). Если у нас было направление и вправо, и влево, то ориентация однозначна, а иначе у нас все фонари направлены в одну сторону, и, чтобы можно было развернуть, надо, чтобы все s_i были одинаковыми.

Тогда посчитаем ответ следующим образом: посчитаем dp аналогично второй подгруппе, это найдёт количество способов начать с левой границей в x. Посчитаем аналогичное dp в обратном порядке по правым границам, тогда мы найдём количество способов закончить в x.

Теперь переберём точку, где соприкасаются префикс и суффикс, и перемножим значения dp. Не забудем прибавить все отдельные значения dp, чтобы учесть направления только в одну сторону.

Осталось вычесть из ответа n, так как мы два раза учли подмножество из одного фонаря.

Получаем решение за $O(n + X)$, где X — максимальное ограничение на координаты.

Подгруппа 4: $t = 1$, $s_i = s_j$

Аналогично предыдущей подгруппе посчитаем две dp. Теперь поймём, как устроен ответ: это либо все направлены в одну сторону, тогда надо просуммировать значения любой из dp, либо у нас есть фонари направленные и вправо, и влево, тогда надо перебрать точку соприкосновения и перемножить.

Получаем решение за такую же асимптотику.

Подгруппа 5: $t = 1$, $n \leqslant 1000$

Напишем $dp_{i,x}$ — количество способов выбрать подмножество экономных фонарей вместе с их ориентацией, если мы рассмотрели только первые i фонарей, а последняя освещённая координата x. Тогда переходы в этой динамике следующие:

- $dp_{i+1,x_i} = dp_{i,x_i-s_i} + 1$. Мы направли фонарь i влево.
- $dp_{i+1,x_i+s_i} = dp_{i,x_i} + 1$. Мы направли фонарь i вправо.

Осталось понять, какие состояния мы учли несколько раз. Экономное множество фонарей может учитываться несколько раз, если каждый из фонарей можно ориентировать в другую сторону, и подмножество останется экономным. Очевидно, что в таком множестве все фонари будут направлены в одну сторону, поскольку иначе будет разрыв. Тогда $i_1, i_2, \ldots, i_k$ можно ориентировать как влево, так и вправо, если выполняется $x_{i_1} + s_{i_1} = x_{i_2}, x_{i_2} + s_{i_2} = x_{i_3}, \ldots, x_{i_{k-1}} + s_{i_{k-1}} = x_{i_k}$ и $s_{i_1} = s_{i_2} = \ldots = s_{i_k}$.

Тогда ответ на задачу будет суммой всех значений dp минус количество таких подмножеств. Динамика считается за $O(n^2)$ или за $O(n^2 \cdot log(n))$, если динамику хранить в `std::map`. А количество таких множеств тоже считается за $O(n^2)$, так как максимальная такая «цепочка» имеет длину n, и мы можем её явно перебрать.

Подгруппа 6: $t = 1$

Осталось научиться считать за линейное время dp, а также количество «цепочек». Для пересчёта динамики можно заметить, что для очередного i меняются только 2 состояния по x. Поэтому можно хранить dp_x — количество экономных подмножеств, оканчивающихся в точке x. И для очередного i пересчитать следующим образом все состояния:

- $dp_{x_i+s_i} := dp_{x_i+s_i} + dp_{x_i} + 1$.
- $dp_{x_i} := dp_{x_i} + dp_{x_i-s_i} + 1$.

Для подсчёта цепочек можно написать dp_i^2 — количество «цепочек», которые кончаются на i-м фонаре. Тогда для того, чтобы пересчитать количество «цепочек», оканчивающихся на j-м фонаре, достаточно проверить, есть ли фонарь в координате $x_j - s_j$, и, если он есть (пусть его номер i), тогда, если $s_i = s_j$, то $dp_j^2 = dp_i^2 + 1$, иначе $dp_j^2 = 1$.

Подгруппа 7: $t = 2$, если $x_i = x_{i+1}$, то $s_i \neq s_{i+1}$

Также посчитаем динамику из подгруппы 6. Тогда если у нас в очередной координате ровно один фонарь, то переходы остаются такими же, иначе они следующие. Пусть номер фонаря, для которого мы пересчитываем состояния, равен i, первый из его прожекторов имеет силу освещения s_1, а второй s_2. Тогда:

- $dp_{x_i+s_1} := dp_{x_i+s_1} + dp_{x_i} + dp_{x_i-s_2} + 2$. Первый прожектор ориентировали вправо.
- $dp_{x_i+s_2} := dp_{x_i+s_2} + dp_{x_i} + dp_{x_i-s_1} + 2$. Второй прожектор ориентировали вправо.
- $dp_{x_i} := dp_{x_i} + dp_{x_i-s_1} + dp_{x_i-s_2} + 2$. Ровно один из прожекторов ориентировали влево.

Осталось также посчитать, сколько есть подмножеств, у которых можно поменять ориентацию, чтобы подмножество осталось экономным. Но при двух прожекторах на одном фонаре у нас может быть более сложная конструкция: мы можем поменять между собой направления двух прожекторов местами. Поэтому для учёта «цепочек» мы можем написать $dpchain_{x,y}$ — количество «цепочек», которые заканчиваются в координате x, и, если мы поменяем ориентацию прожекторов (если один прожектор светил влево, другой вправо, то после смены ориентации будет наоборот) у последнего фонаря, то последняя освещённая координата увеличится на y. Тогда в ней переходы следующие в случае (мы говорим, что $s_1 > s_2$):

- $dpchain_{x_i,s_1} := dpchain_{x_i,s_1} + 1 + dpchain_{x_i-s_1,s_1}$. Мы в фонаре поставили прожектор s_1, который светит влево.
- $dpchain_{x_i,s_2} := dpchain_{x_i,s_2} + 1 + dpchain_{x_i-s_2,s_2}$. Мы в фонаре поставили прожектор s_1, который светит вправо.
- $dpchain_{x_i+s_2,s_1-s_2} := dpchain_{x_i+s_2,s_1-s_2} + 1 + dpchain_{x_i-s_1,s_1-s_2}$. Мы в фонаре поставили прожектор s_2, который светит вправо, и прожектор s_1, который светит влево. Тогда, если мы поменяем ориентацию двух фонарей, то станет покрыта координата $x_i + s_1$.

В случае, если у фонаря ровно один прожектор, то переходы будут следующие:

- $dpchain_{x_i,s_i} := dpchain_{x_i,s_i} + 1 + dpchain_{x_i-s_i,s_i}$.

Эту динамику можно хранить в `std::map<int, map<int, int>`. Тогда заметим, что все «цепочки» были посчитаны в этой динамике, так как, если мы меняем местами прожекторы (для лучшего понимания можно считать, что, если был поставлен один прожектор, то было поставлено два прожектора, один из которых освещает отрезок длины 0), то каждый отрезок, освещённый одним фонарём, сдвигается на дельту сил прожекторов. А значит, ответ — сумма всех состояний dp минус сумма всех состояний $dpchain$.

Полное решение:

В последней подгруппе требуется понять, что нужно немного поменять переходы:

- $dp_{x_i+s_i} := dp_{x_i+s_i} + 2 \cdot dp_{x_i} + dp_{x_i} + 3$.
- $dp_{x_i} := dp_{x_i-s_i} + 2 \cdot dp_{x_i} + dp_{x_i} + 2$.

А $dpchain$ пересчитывается следующим образом:
$dpchain_{x_i,s_i} := dpchain_{x_i,s_i} + 2 + 2 \cdot dpchain_{x_i-s_i,s_i}$.

Задача 7. Марсианский рюкзак

Переформулировка задачи

Для каждого префикса из первых i предметов нужно найти минимально возможное значение побитового `OR` выбранного подмножества, у которого суммарная стоимость не меньше C. Если такого подмножества нет, ответ равен -1.

Подзадача 1: $n \leqslant 20$, $k \leqslant 10$

Можно просто перебрать все подмножества первых i предметов для каждого i. Для каждого подмножества считаем сумму стоимостей и побитовое `OR` странностей. Среди подмножеств с суммой не меньше C берём минимальный `OR`.

Асимптотика $\mathcal{O}(2^0 + 2^1 + \ldots + 2^n) = \mathcal{O}(2^n)$.

Подзадачи 2 – 3: $n \leqslant 50\,000$, $k \leqslant 10$

Для фиксированной маски x максимальная суммарная стоимость подмножества с общей странностью $y \subseteq x$ равна $F_i(x) = \sum_{j \leqslant i,\ w_j \subseteq x} c_j$, где запись $w_j \subseteq x$ означает, что w_j является подмаской x, то есть $(w_j \& \sim x) = 0$.

Значит, ответ для префикса i — это минимальная маска x, для которой $F_i(x) \geqslant C$.

Будем поддерживать значения $F_i(x)$ напрямую: когда приходит предмет с маской w_i и стоимостью c_i, нужно прибавить c_i ко всем маскам x, для которых $w_i \subseteq x$. После этого достаточно найти минимальную маску x с $F_i(x) \geqslant C$.

Асимптотика $\mathcal{O}\left(n \cdot 2^k\right)$.

Подзадачи 4 – 5: $n \leqslant 1\,000\,000$, $k \leqslant 19$, w_i — **степени двойки или** $n \leqslant 2000$

В данных подзадачах, чтобы найти нужный ответ для префикса p, можно идти жадно по битам сверху вниз. Пусть вначале в маске разрешены все биты, а текущая доступная суммарная стоимость равна $T = \sum_{i=1}^{p} c_i$. Далее идём по битам от старших к младшим и пытаемся удалить текущий бит, за $\mathcal{O}(p)$ находим T' – новую сумму c_i по подходящим предметам. Если $T' < C$, то возвращаем бит, иначе оставляем его нулем. Далее переходим к меньшему биту.

Асимптотика $\mathcal{O}(n^2 k)$, что достаточно для решения подзадачи 4.

Для решения подзадачи 5 можно просимулировать алгоритм выше, заметив, что T' можно вычислить как вычитание из T суммы c_i с w_i, равными нужной степени двойки.

Подзадача 6: $n \leqslant 500\,000$, $k \leqslant 16$

Для данной подзадачи нужно применить технику «Meet In The Middle».

Каждую маску w_i представим в виде $w_i = (h_i, \ell_i)$, где:

- h_i — старшие p бит,
- ℓ_i — младшие q бит.

Аналогично ответ будем искать в виде $ans_i = (H_i, L_i)$, где сначала минимизируем старшую часть H_i, а затем при фиксированной H_i минимизируем младшую часть L_i.

Для нахождения старшей части применяем решение из подзадачи 3 за $\mathcal{O}(n \cdot 2^p)$. Тогда при добавлении нового предмета H_i либо не изменяется (остается равным H_{i-1}), либо уменьшается.

Если $H_i < H_{i-1}$, то пересчитаем значение функции $F_i(x)$ для всех $H_i \cdot 2^q \leqslant x < (H_i + 1) \cdot 2^q$. Для этого по всем j, таким, что $1 \leqslant j \leqslant i$, и если $h_j \subseteq H_i$, то в $F_i(H_i \cdot 2^q + l_j)$ добавим c_j, а далее сделаем SOS-DP для F в промежутке $[H_i \cdot 2^q, (H_i + 1) \cdot 2^q)$.

Иначе, если $H_i = H_{i-1}$ и $h_i \subseteq H_i$, то обновим нужные значения F за $\mathcal{O}(2^q)$.

Итого, так как уменьшений H не более $\mathcal{O}(2^p)$, то решение суммарно работает за $\mathcal{O}(2^k \cdot q + n \cdot (2^p + 2^q))$. При $p = q = 2^{\frac{k}{2}}$ решение работает за $\mathcal{O}(2^k \cdot k + n \cdot 2^{\frac{k}{2}})$.

Полное решение

Будем строить ответы сразу для целого отрезка префиксов и сразу по всем битам. Пусть старшие биты ответа уже зафиксированы, а осталось определить ещё t младших бит.

Для текущего рекурсивного вызова будем хранить:

- число `pref` — уже зафиксированная часть ответа в старших битах;
- отрезок префиксов $[L, R]$, для которых все ответы имеют один и тот же уже построенный старший префикс;
- массив cnt длины 2^t.

Массив cnt построен по первым $L - 1$ предметам. Для каждой маски s длины t значение $cnt[s]$ равно суммарной стоимости тех предметов, которые:

- среди уже обработанных старших битов не противоречат маске `pref`, то есть их старшие биты являются подмаской соответствующих битов ответа;
- на оставшихся t младших битах точно равны s.

Иначе говоря, мы сгруппировали все предметы, которые ещё могут входить в ответ, по их оставшейся младшей части.

Как определить очередной бит? Пусть сейчас мы определяем старший из оставшихся t бит, то есть бит с номером $t-1$ внутри текущего массива cnt.

Разобьём все маски на две половины:

- нижняя половина — маски, у которых этот бит равен 0;
- верхняя половина — маски, у которых этот бит равен 1.

Если мы хотим поставить текущий бит ответа в 0, то брать можно только предметы из нижней половины. Максимальная суммарная стоимость таких предметов равна $S_0 = \sum_{s<2^{t-1}} cnt[s]$.

Отсюда сразу следует критерий:

- если $S_0 \geqslant C$, то существует допустимое решение с текущим битом 0;
- если $S_0 < C$, то никакое решение с текущим битом 0 невозможно, значит, этот бит обязан быть равен 1.

Этого достаточно, так как, если текущий бит ответа равен 0, то остальные более младшие биты мы в лучшем случае можем поставить в 1. Значит, мы действительно учитываем все предметы, которые можно взять при текущем бите 0.

Где разбивается отрезок префиксов? Из-за монотонности ответов существует такая граница M, что

- для префиксов $L, L+1, \ldots, M-1$ текущий бит ответа равен 1;
- для префиксов $M, M+1, \ldots, R$ текущий бит ответа равен 0.

Найдём её одним проходом.

Изначально массив cnt уже соответствует первым $L-1$ предметам, а значение $S_0 = \sum_{s<2^{t-1}} cnt[s]$ соответствует тем из них, которые можно брать при текущем бите 0.

Теперь будем последовательно рассматривать предметы с номерами $L, L+1, \ldots, R$. Для очередного предмета i проверим, может ли он участвовать в решении текущего узла:

- на уже зафиксированных старших битах его маска не должна содержать единиц там, где в `pref` стоит ноль;
- текущий рассматриваемый бит у него должен быть равен нулю, если мы хотим считать сумму S_0.

Как только после добавления очередного предмета можно впервые получить $S_0 \geqslant C$, мы нашли позицию M.

Время поиска границы на одном узле равно $\mathcal{O}(R-L+1)$.

После того как граница M найдена, остаются две рекурсии.

Левая часть рекурсии отвечает за префиксы $[L, M-1]$. Теперь текущий бит уже зафиксирован как единица, значит, на этом бите предмет может иметь и 0, и 1. Следовательно, для каждой маски младших $t-1$ бит нужно просто сложить две половины: $next_1[s] = cnt[s] + cnt[s+2^{t-1}]$ для $0 \leqslant s < 2^{t-1}$.

Стоимость построения массива для левой части: $\mathcal{O}(2^t)$.

Правая часть рекурсии отвечает за префиксы $[M, R]$. В этом случае брать можно только нижнюю половину: $next_0[s] = cnt[s]$ для $0 \leqslant s < 2^{t-1}$.

Но есть важный нюанс: массив для правого сына должен соответствовать уже первым $M-1$ предметам, а массив cnt был построен только для первых $L-1$.

Поэтому во время поиска границы M мы одновременно поддерживаем и массив $next_0$: все подходящие предметы с текущим битом 0, встретившиеся на позициях от L до $M-1$, добавляются в нужную ячейку по своим младшим $t-1$ битам.

Тогда к моменту запуска правой рекурсии массив уже готов.

Остаётся отдельно обработать префиксы, для которых ответа вообще не существует. Это делается заранее: найдём первый префикс, у которого суммарная стоимость всех предметов стала хотя бы C. Для всех предыдущих префиксов выводим -1, а рекурсию запускаем только с этого места.

Итого, решение за $\mathcal{O}(nk + 2^k k)$.

Задача 8. Хорошие раскраски — 8

Подзадача 1.

В случае $n = 3$ ответ равен $k(k-1)$, если все рёбра активны и k в ином случае.

Подзадача 2.

Пусть Γ_i $(1 \leqslant i < m)$ — это грань, содержащая на своей границе ребро $l_i \to l_{i+1}$, а Γ_0 — это грань, содержащая на своей границе ребро $l_m \to l_1$.

Тогда можно сначала раскрасить грань Γ_0, на это есть k способов, затем останется $(k-1)$ свободных цветов для грани Γ_1, а если продолжить процесс раскраски граней в порядке $\Gamma_2, \ldots, \Gamma_m$ далее, то для каждой очередной грани будет доступно $(k-2)$ цветов.

Таким образом, в зависимости от n ответ на задачу будет равен:

$$\begin{aligned} &\text{answer} = k, && n = 3 \\ &\text{answer} = k(k-1)(k-2)^{\frac{n-5}{2}}, && n \geqslant 5 \end{aligned}$$

Подзадача 3.

В этом случае, если все рёбра не активны, то ответ равен k, а в ином случае, если будет активно ровно x рёбер, то грани объединятся в x граней, которые соединены по циклу друг с другом. Количество способов раскрасить x таких граней в k цветов можно вычислить с помощью метода динамического программирования:

- $\mathrm{dp}_1[i]$ — количество способов раскрасить i граней в k цветов так, чтобы любые две грани i и $i+1$ имели различные цвета, а также чтобы первая и последняя грань имели различные цвета.
- $\mathrm{dp}_2[i]$ — количество способов раскрасить i граней в k цветов так, чтобы любые две грани i и $i+1$ имели различные цвета, но при этом первая и последняя грани должны иметь один и тот же цвет.

Очевиден базовый случай $\mathrm{dp}_1[1] = 0$, $\mathrm{dp}_2[2] = k$. А переходы расписываются так:

$$\begin{aligned} \mathrm{dp}_1[i] &= \mathrm{dp}_1[i-1] \cdot (k-2) + \mathrm{dp}_2[i-1] \cdot (k-1) \\ \mathrm{dp}_2[i] &= \mathrm{dp}_1[i-1] \end{aligned}$$

Подзадачи 4–5.

В эти подзадачи заходили переборные решения. Например, авторское решение имеет время работы $O((q+1) \cdot (n^3 + k^n \cdot n))$ и устроено следующим образом:

- Явно строится граф соседних граней в исходном графе.
- С помощью алгоритма Флойда-Уоршелла грани разбиваются на группы, которые должны объединиться в одну грань.
- Перебираются всевозможные k^n раскрасок исходных граней и явно проверяются на корректность.

Подзадача 6.

Для решения этой подзадачи достаточно было заметить, что ответ равен 0, если в дереве есть вершина нечётной степени, и равен 2 во всех остальных случаях.

Подзадача 7.

Замечание. Здесь и далее мы будем обозначать $\chi(n)$ — как количество способов раскрасить n граней, соединённых по циклу, используя k цветов.

Совместим идеи из 2-й и 3-й подзадач: сначала раскрасим грань Γ_0, которая *накрывает* дерево сверху, а затем будем раскрашивать грани в порядке их вложенности. Причём грани, ограниченные сверху одной и той же вершиной дерева, будем раскрашивать одновременно.

Если вершина v является корнем дерева, то требуется раскрасить $\deg v - 1$ граней, это можно сделать $\frac{\chi(\deg v)}{k}$ способами. Если же v не является корнем, то под ней находится $\deg v - 2$ граней, и их можно раскрасить $\frac{\chi(\deg v)}{k(k-1)}$ способами. Таким образом, требуется перемножить эти величины по всем нелистовым вершинам v, полученное значение и будет ответом на задачу.

Подзадачи 8–10.

Назовём вершину v *висячей*, если она не является листом и соединена не более чем одним активным ребром с другими вершинами. Заметим, что от удаления *висячей* вершины ответ не изменяется, поэтому идея решения этих подзадач заключается в том, чтобы отвечать на запросы независимо и наиболее эффективным образом уметь избавляться от таких вершин.

Авторское решение предлагает воспользоваться тем, что в задаче явно задан массив предков $p_i < i$, а поэтому избавиться от *висячих* вершин можно в два прохода:

- Сначала перебираем вершины в порядке от n до 1, если вершина v является листом, то она не висячая и она не удаляется. В ином случае вершина удаляется только тогда, когда в неё не ведут активные рёбра из неудалённых вершин с большими номерами.
- Во втором проходе будем рассматривать вершины v в порядке от 1 до n. Если вершина v является корнем своей компоненты (то есть из неё наверх не исходит активное ребро в неудалённую вершину) и при этом эта вершина имеет единственного сына, то такая вершина удаляется.

После двух таких проходов в дереве не остаётся висячих вершин. Но, возможно, дерево распадается на компоненты связности. Заметим, что мы всё также можем раскрашивать грани в порядке вложенности, и каждая вершина v, которая является корнем, будет давать множитель $\frac{\deg' v}{k}$ к ответу, а все остальные нелистовые вершины будут давать множитель $\frac{\deg' v}{k(k-1)}$.

Представленное решение имеет асимптотику $O(n \cdot q)$, а также быстро работает, так как отсутствуют прыжки по памяти и вся работа, по сути, совершается в двух-трёх циклах `for`.

Подзадача 11.

В этой подзадаче предлагается сжать исходное дерево до размера $4q$ так, чтобы ответ от этого не изменился с точностью до домножения на константный множитель M. Делать это достаточно сложно, так как требуется проверять достаточно много случаев, поэтому при самостоятельной реализации этой идеи могут потребоваться стресс-тесты.

Общая идея заключается в том, чтобы разбить все рёбра на два класса: *постоянные* и *нестабильные*. Постоянными мы назовём рёбра, которые активны на протяжении всех запросов, нестабильными те рёбра, которые изменяют своё состояние в одном из запросов.

Заведём массив e, где $e(v)$ — количество вершин на внешнем цикле, которые соединены постоянными рёбрами с v. Для инициализации положим, что $e(v) = 1$ для листьев v и $e(v) = 0$ для всех остальных вершин. Соответственно, мы перестаём считать, что цикл проходит именно через листья дерева, а предполагаем, что он проходит через какие-то другие (назовём их *виртуальными*) вершины, которые учтены в значениях $e(v)$. Ясно, что от этого ответ на задачу не изменится.

Далее вычислим для каждой вершины v величину:

- $\min f(v)$ — максимальное количество непересекающихся путей, которые ведут вниз по дереву от вершины v в виртуальные вершины по постоянным рёбрам.

Тогда можно сжать дерево, используя следующие манипуляции:

- Выделить цепочки вершин вида $v_1 \to v_2 \to \ldots \to v_l$, где каждая пара соседних вершин в цепочке соединена ребром и вершины $v_2, \ldots, v_{l-1}$ имеют ровно двух соседей. Тогда такие цепочки можно объединить в одно ребро, соответствующим образом модифицировав последовательность запросов — новое ребро будет активным тогда и только тогда, когда активны все рёбра из исходной цепочки.

Затем следует выделить *опорные* вершины, то есть те вершины v, для которых выполнено $\min f(v) \geqslant 2$. Все вершины v, из которых по рёбрам, ведущим наверх, достижима какая-то отличная от v опорная вершина и для которых выполнено $\min f(v) \geqslant 1$, будем называть *предсказуемыми*. Такие вершины интересны тем, что они никогда не будут удалены в ходе последовательного удаления висячих вершин.

Поэтому каждую предсказуемую вершину можно переподвесить к виртуальному корню -1, а в её бывшем предке p увеличить значение $e(p)$ на единицу, это никак не изменит ответ на задачу. Вершину -1 для удобства мы будем считать опорной (но в ответе для неё множитель считать не будем, так как она не является реальной вершиной, а выступает удобной записью для факта, что нам не важно, куда именно подвешена вершина v). Если вершина v подвешена за вершину -1 и $\min f(v)$ совпадает с количеством её сыновей, то такая вершина будет давать постоянный вклад в ответ, который можно учесть в множителе M, после чего, удалив эту вершину из дерева, всех её сыновей переподвесить к вершине -1.

Подробный анализ показывает, что после сжатия дерева таким образом останется не более $4q$ вершин. Таким образом, решить эту подзадачу можно за время $O(n + q^2)$.

Подзадача 12.

Попробуем оптимизировать решение за $O(nq)$ другим способом, воспользовавшись тем, что высота дерева h не превосходит 20. Идея решения заключается в том, чтобы явно поддерживать дерево, которое остаётся после удаления висячих вершин (см. решение подзадач 8–10). Для этого достаточно разделить вершины на три типа:

- N — вершины, удалённые после первого прохода;
- S — вершины, удалённые после второго прохода;
- P — вершины, которые не будут удалены в ходе алгоритма.

Теперь рассмотрим процесс удаления ребра $v \to p_v$. Сначала следует пробежаться по активным рёбрам вверх по вершинам, начав с вершины p_v, до того момента, пока в очередную вершину не будет входить ещё какое-то активное ребро снизу, соединяющее её с вершиной P-класса, либо пока мы не дойдём до корня дерева. Все посещённые вершины перейдут в класс N.

Если мы остановились в вершине, в которую ведёт ещё одно ребро, то в случае, если это ребро единственное, следует дальше продолжать подниматься вверх по активным рёбрам. Если мы остановились из-за того, что из очередной вершины u не исходит активного ребра вверх, то требуется спуститься вниз от вершины u до первой вершины, которая будет листом или у которой будет два сына P, соединённых с ней активными рёбрами. Все посещённые вершины следует отметить классом S. Если же мы дошли до вершины, в которую ведёт ещё одно активное ребро из вершины типа P, то не требуется дополнительно изменять классы вершин.

Так как мы изменяем классы у вершин по одной, то можно с лёгкостью пересчитывать ответ.

Данный алгоритм работает за время $O(n + q \cdot h)$.

Подзадачи 13–15.

Есть два подхода для решения задачи на полный балл, оба работают за время $O(n + q \log n)$, но основаны на разных принципах.

Подход 1. (D&Q) Заметим, что можно слегка модифицировать решение подзадачи 11 (требуется дополнительно обрезать из каждой компоненты стабильное или нестабильное ребро, ведущее наверх, которое гарантированно состоит из висячих вершин) и применить его в методе разделяй и властвуй.

Сперва сожмём дерево, а затем разделим запросы на две примерно равные половины и решим задачу для этих половинок запросов рекурсивно. То есть будем сжимать дерево под запросы и снова делить их на две равные группы. Так как дерево, соответствующее отрезку из q запросов, по размеру не превышает $O(q)$, то такое решение будет работать за $O(q \log q)$.

Подход 2. (Структуры данных): Основан на том, чтобы поддерживать сжатое дерево на вершинах, оставшихся в P