Задача A. Coffee Cocktail

Автор задачи: Георгий Чарковский, разработчик: Константин Бац

Сразу упростим задачу: информация про общую массу каждого ингредиента и долю кофеина в нем излишняя, можно сразу перейти к рассмотрению массы кофеина в каждом ингредиенте ($\frac{m_i \cdot k_i}{100}$), так как остальная масса нас не интересует. Обозначим эту массу за c_i.

Подзадача 1

В первой подзадаче гарантировалось, что существует не более одного типа снека. Это означает, что если ответ и существует, то он равен 1. Соответственно, достаточно проверить, что всех снеков, взятых вместе, хватает, чтобы набрать x кофеина.

Подзадача 2

Во второй подзадаче, наоборот, все типы снеков различны. При этом для всех снеков m_i и k_i одинаковы, а значит и c_i одинаковы и равны m_i. Следовательно, чтобы набрать x кофеина, понадобится хотя бы $\left\lceil \frac{x}{m_i} \right\rceil$ снеков, и это число и будет ответом (либо -1, если n снеков не хватит).

Подзадача 3

Решение третьей подзадачи уже приближает нас к полному решению. Если все k_i равны, а t_i различны, то ответом будет просто количество различных взятых снеков. При этом, чтобы набрать x кофеина минимальным числом снеков, достаточно выбирать снеки с максимальным содержанием кофеина, что, в данном случае, соответствует максимальным m_i. То есть достаточно отсортировать все снеки по убыванию m_i и выбирать в таком порядке, пока не наберется x кофеина или пока снеки не закончатся. Время работы решения — $\mathcal{O}(n \log n)$.

Подзадача 4

Аналогично предыдущей подзадаче, но уже следует в явном виде вычислять c_i и сортировать по убыванию c_i, после чего набирать в таком порядке. Стоило обойтись вычислениями в целых числах и использовать $100 \cdot c_i$, чтобы набрать $100 \cdot x$ кофеина, однако в ограничениях данной задачи точности `double` тоже должно было быть достаточно.

Подзадачи 5 и 6

Последняя идея, необходимая для полного решения — группировка по типам снеков. Если все k_i равны, то наиболее выгодно сначала набирать снеки максимальной массы. Но если у нас есть один снек массы 100 одного типа и два снека массами по 51 другого типа, то выгодно начать с них. Действительно, как только мы выберем один снек какого-то типа, нет смысла не брать все другие снеки того же типа, так как мы приблизимся к цели, а ответ не увеличится.

Таким образом:

1. Посчитаем для каждого типа снека от 1 до q суммарную массу кофеина в снеках этого типа. В подзадаче 5 для этого достаточно посчитать суммарную массу и умножить на одинаковый $\frac{k_i}{100}$.
2. Упорядочим типы по убыванию суммы c_i.
3. Будем набирать кофеин в этом порядке, пока не наберем x или пока типы не закончатся.
4. Использованное количество типов и будет ответом.

Время работы решения все еще $\mathcal{O}(n \log n)$.

Задача B. Fraction Conversion

Автор задачи и разработчик: Егор Юлин

Для начала сделаем базовые наблюдения для всех подзадач:

- a не влияет на ответ — целую часть можно представить в любой системе счисления;
- $0.b(c)$ можно записать как $\frac{b}{10^m} + \frac{c}{10^{m+k}} + \frac{c}{10^{m+2k}} + \ldots$;
- если число представляется в виде обыкновенной дроби $\frac{x}{y}$, то ответом на задачу будет являться такое минимальное z, что z^k делится на y для некоторого целого k.

Подробнее про последний пункт: мы хотим представить число в виде суммы слагаемых вида $\frac{t_i}{c^i}$ по целым i. При чем, если сумма конечная, то можно все слагаемые привести к общему знаменателю по минимальному вошедшему i: например, $\frac{3}{5^4} + \frac{1}{5^2} + \frac{2}{5} = \frac{3+5^2+2\cdot5^3}{5^4}$. Соответственно, обратное тоже верно: если дробь записывается в виде $\frac{x}{z^k}$, то ее можно представить в СС с основанием z, разложив эту дробь на соответствующие слагаемые.

Таким образом, если мы переведем данную нам дробь в каноничный и несократимый вид $\frac{x}{y}$ и разложим y на простые как $y = p_q^{\alpha_1} \cdot \ldots \cdot p_t^{\alpha_t}$, то ответом будет $c = p_1 \cdot p_2 \cdot \ldots \cdot p_t$, так как $\frac{x}{y}$ умножением на одно и то же число и числителя, и знаменателя, можно привести к виду $\frac{x'}{c^{\max \alpha}}$.

Подзадача 1

Тем не менее, для решения первых подзадач не обязательно было проводить такое количество рассуждений, хоть они и будут для нас полезны. В случае $m = 0$, $k = 1$ есть всего 10 возможных вариантов того, как будет выглядеть дробь — от $0.(0)$ до $0.(9)$. Для каждой из них можно просто вручную посчитать ответ: $0.(0) = 0$, ответ — 1, $0.(1) = \frac{1}{9}$, что представляется как 0.01 в троичной системе (ответ — 3), остальные также дадут ответ 3, кроме $0.(9) = 1$, для которого ответ равен 1.

Подзадача 2

В случае, когда $m = 1$ и $k = 1$, дробь представляется как $\frac{b}{10} + \frac{c}{90}$. Иными словами, $\frac{9b+c}{90}$. Если сократить числитель и знаменатель, дальше можно для каждого возможного знаменателя в явном виде выписать ответ, либо же в программе разложить полученный знаменатель на простые множители и перемножить: ответом будет произведение какого-то подмножества элементов из $\{2, 3, 5\}$.

Подзадача 3

В подзадаче 3 гарантировалось, что у дроби нет периода. В таком случае ее перевод в обыкновенную дробь — это просто поиск $\texttt{gcd}(b, 10^m)$, и, более того, нам уже известно, что ответ $c \leqslant 10$, потому что в десятичной системе она представима без периода.

Достаточно проверить, делится ли b на 2^m и на 5^m, и если да, то соответствующее простое сократится и из числителя, и из знаменателя, и не войдет в качестве множителя в ответ. Например, второй пример — 0.25 представляет из себя $\frac{25}{100}$ или $\frac{1}{4}$, что представляется как 0.01_2.

Подзадача 4

Аналогичные рассуждения работают и здесь, но надо только упростить бесконечную сумму. Упростим ее сразу в общем виде: $\frac{b}{10^m} + \frac{c}{10^{m+k}} + \frac{c}{10^{m+2k}} + \ldots$ содержит геометрическую прогрессию со знаменателем $\frac{1}{10^k}$, а значит можно переписать эту сумму как

$$\frac{b}{10^m} + \frac{c \cdot 10^k}{10^m \cdot (10^k - 1)}.$$

В этой подзадаче $m = 0$, поэтому b отсутствует, и задача тоже сводится к поиску $\texttt{gcd}(c \cdot 10^k, 10^k - 1)$ и разложению $\frac{10^k-1}{\texttt{gcd}(c \cdot 10^k, 10^k-1)}$ на простые. Поскольку 10^k и $10^k - 1$ всегда взаимно просты, то достаточно найти $\texttt{gcd}(\texttt{c}, \texttt{10}^\texttt{k} - \texttt{1})$, после чего за $\mathcal{O}(sqrt10^k - 1)$ разложить оставшийся знаменатель на простые.

Подзадача 5

В случае, когда $m + k \leqslant 12$, $\frac{b}{10^m} + \frac{c \cdot 10^k}{10^m \cdot (10^k-1)}$, которое можно переписать как

$$\frac{(10^k - 1)b + 10^k c}{10^m \cdot (10^k - 1)}$$

имеет и числитель, и знаменатель до 10^{12}. Следовательно, с тем же подходом остается найти их $\texttt{gcd}$, поделить числитель и знаменатель на него, и оставшийся знаменатель разложить на простые, что укладывается в ограчения по времени.

Подзадача 6

Для полного решения необходимо воспользоваться небольшим трюком. Сразу разложим знаменатель на простые числа: $10^m \cdot (10^k - 1)$ содержит 2^m, 5^m и разложение $10^k - 1$, которое можно найти за $\mathcal{O}(\sqrt{10^k})$. Осталось определить, какие из этих простых входят в числитель и сократятся.

Для этого сразу заметим, что сумма двух чисел всегда делится на меньшую из степеней простого, в которых оно входит в слагаемые (например, $4 + 32$ делится на 2^2, но не на 2^3). А тогда:

- 2 и 5 входят в $10^k c$ в степени $k + \texttt{deg}(c)$, где $\texttt{deg}(c)$ — степень их вхождения в c, которая может быть найдена за $\mathcal{O}(\log c)$;
- в $(10^k - 1)b$ же они входят ровно в степени $\texttt{deg}(b)$, потому что $10^k - 1$ не делится ни на 2, ни на 5;
- простые из разложения $10^k - 1$ же, наоборот, не входят в 10^k, поэтому в числителе $(10^k - 1)b + 10^k c$ степень их вхождения равна $\texttt{deg}(c)$ по описанным выше причинам.

Можно для каждого простого из разложения $10^k - 1$ независимо найти его степень вхождения в c, а можно просто сразу посчитать $\texttt{gcd}(10^k - 1, c)$. После остается только сократить для каждого простого его вхождения в числитель и знаменатель, и перемножить те простые, которые остались в знаменателе в ненулевой степени. Полученное произведение и будет ответом.

Задача C. Public Transportation

Авторы задачи: Даниил Орешников и Георгий Чарковский, разработчик: Даниил Орешников

Подзадача 1

Посмотрим на матрицу, в которой $1 \leqslant t_{i,j} \leqslant 2$. Заметим сразу, что чтобы треугольник был хороший, в его прямом углу должно стоять хотя бы число 2 (т.к. $a > 0$ и $b > 0$). При этом сумма двух остальных углов должна быть равна прямому. Если прибавить к матрице $d < 0$, то не найдется подходящего кандидата на то, чтобы быть прямым углом. Если прибавить $d > 0$, то сумма двух острых углов будет хотя бы $2(1 + d)$, что гарантированно больше максимально возможного значения в прямом углу $2 + d$.

Таким образом, достаточно просто посчитать количество хороших треугольников в заданной матрице. А такие треугольники имеют вид $t_{i,j} = 2$, $t_{i+1,j} = t_{i,j+1} = 1$. Количество двоек, рядом с которыми справа и снизу стоят единицы, можно посчитать проходом по матрице за время $\mathcal{O}(nm)$.

Подзадача 2

Каждый треугольник при $n = 2$ обязан иметь прямой угол в первой строке, а один из острых — в том же столбце на строку ниже. Переберем за $\mathcal{O}(m^2)$ все возможные пары углов в первой строке и проверим, может ли соответствующий треугольник быть хорошим.

Пусть мы рассматриваем $t_{1,j} = x$, $t_{2,j} = y$ и $t_{1,k} = z$. Тогда нам необходимо добавить такое d, чтобы $t_{1,k}$ стало равно противоположной стороне, то есть 1. Иными словами, единственное подходящее d — это $1 - t_{1,k}$. Добавим его к $t_{1,j}$ и $t_{2,j}$, и проверим, что оставшиеся условия выполняются.

Подзадача 3

Аналогично первой подзадаче, заметим, что имеет смысл рассматривать только d от $-\max(T)+2$ до $\max(T)$. Если прибавить меньшее значение, не останется чисел $\geqslant 2$. Если прибавить большее — сумма любых двух клеток будет больше любой другой.

Переберем d от -50 до 50, а после за $\mathcal{O}(n^2m^2)$ переберем все возможные треугольники и проверим их на «хорошесть». Такое решение при эффективной реализации проходит тесты третьей подзадачи.

Чтобы гарантированно пройти эту подзадачу, следует заметить, что каждый выбранный треугольник может быть хорошим не более, чем для одного конкретного d. Действительно, если одно d подошло, то любое другое нарушит равенство между значением в прямом углу и суммой значений в острых. При этом потенциально подходящее d можно посчитать по формуле. Если в прямом углу стоит x, а в острых — y и z, то нам необходимо, чтобы выполнялось $x + d = (y + d) + (z + d)$, из чего $d = x - y - z$. Осталось перебрать все треугольники за $\mathcal{O}(n^2m^2)$.

Подзадача 5

Если еще немного соптимизировать перебор, можно было пройти подзадачу 5. Пусть $t_{i,j} = x$, а $t_{i+a,j} = y$. Заметим, что значение y по условию должно быть ровно на a меньше значения x. Если это не выполняется, остановим перебор и перейдем к следующим значениям.

В случайно сгенерированной таблице найдется мало пар значений с данным свойством, поэтому перебор за $\mathcal{O}(n^2m^2)$ сделает на самом деле порядка $\mathcal{O}(n^2m)$ действий, так как для большого количества пар клеток третий угол треугольника даже не будет перебираться.

Подзадачи 4 и 6

Еще улучшим описанную выше идею. Заметим, что для всех трех клеток хорошего треугольника выполняется $\texttt{row} + \texttt{col} + \texttt{value} = \texttt{const}$. Иными словами, сумма значения, номера строки и номера столбца одинакова и равна $i + j + a + b$. Таким образом, достаточно только рассматривать группы клеток, у которых эта величина принимает одно и то же значение.

Остается только заметить, что для любого треугольника, у которого все три клетки-вершины имеют одинаковое значение данной величины, найдется единственное подходящее d, чтобы треугольник стал хорошим. Тогда для решения четвертой подзадачи достаточно для каждой клетки независимо перебирать клетки с тем же значением в строке и клетки с тем же значением в столбце, после чего добавлять к ответу произведения их количеств.

Для полного решения остается только сделать предподсчет с помощью хеш-таблицы и найти для каждого возможного значения $t_{i,j} + i + j$ количество клеток с таким значением в каждой строке и каждом столбце. После чего достаточно пройтись по всем клеткам и для каждой клетки добавить к ответу количество клеток с тем же значением ниже ее, умноженное на количество клеток с тем же значением правее ее. Время работы полного решения — $\mathcal{O}(nm)$.

Задача D. Restore Permutation

Автор задачи и разработчик: Даниил Орешников

Подзадача 2

Мы пропустим отдельное решение для подзадачи 1, так как почти любое из описанных делее решений обобщается на нее. Для решения подзадачи 2 достаточно напрямую закодировать исходную перестановку, после чего восстановить ее из закодированной строки. Для этого заметим, что для представления чисел от 1 до n достаточно $\lceil \log_2 n \rceil$ бит, то есть в общем случае не больше 20 бит на число. Выпишем по очереди битовые записи каждого элемента перестановки, используя одинаковое число бит, после чего на втором запуске восстановим перестановку целиком.

Подзадача 3

Эта подзадача стоит немного отдельно от других, так как ожидаемое в ней решение не обобщается напрямую на следующие подзадачи. Посчитаем от перестановки полиномиальный хеш, то есть $\left(\sum_{i=1}^{n} x^{i-1} \cdot p_i\right) \bmod M$ для некоторых простых x и M. Запишем в качестве ответа двоичную запись этого хеша.

Заметим, что при обмене местами элементов p_i и p_j хеш перестановки изменяется на $(p_i - p_j) \cdot (x^i - x^j)$. Прочитав перестановку q и посчитанный изначально хеш, посчитаем новый хеш от q, после чего переберем i и j, и проверим, правда ли старый хеш отличается от нового на данную величину.

Поскольку перестановка может иметь длину до 2000, возможных вариантов исхода порядка $4 \cdot 10^6$, так что если хеш считается по модулю $M \approx 10^9$, с достаточно большой вероятностью будет находиться единственная подходящая пара (i, j). Это решение работает за $\mathcal{O}(n^2)$, что не позволяет его применить для перестановок большей длины.

Подзадача 4

Подзадача 4 не имела конкретного авторского решения и была расчитана на то, что участники могут придумать какой-то свой подход, глобально отличающихся от идей, описанных в этом разборе.

Подзадача 5

Немного изменив подход с хешами, можно решить следующую подзадачу. Для этого разобьем всю перестановку на блоки размера примерно 5000 и независимо посчитаем хеш на каждом из них. На втором запуске разобьем перестановку q на те же блоки, посчитаем хеши, и посмотрим, какие блоки изменились.

Если изменился только один блок, можно в нем с помощью предыдущего решения найти пару измененных элементов. Если же изменились два блока, аналогично переберем позицию в одном и втором, и проверим, что их обмен возвращает все хеши к исходным значениям.

Подзадача 6

Второй метод, который можно было применить в этой задаче, похож на *код Хэмминга*. Посчитаем несколько сумм элементов перестановки, чтобы можно было восстановить с некоторой точностью изменившиеся элементы. А именно: для каждого b посчитаем сумму всех элементов на позициях, в битовой записи которых бит номер b равен 1. Для этого предварительно дополним перестановку нулями до длины, равной степени двойки.

Тогда c_0 — сумма по всем позициям, имеющим 1 в нулевом бите, то есть по всем нечетным позициям, c_1 — сумма всех элементов на позициях, имеющих 1 в первом бите, и так далее, $c_{\lceil \log_2 n \rceil}$ — сумма второй половины перестановки. Выведем все $\lceil \log_2 n \rceil$ таких сумм по очереди в качестве битовой строки.

На втором запуске посчитаем такие же суммы для перестановки q. Каждая сумма либо изменилась, либо нет. Если сумма не изменилась, то оба числа, которые поменялись местами, либо входили в сумму, либо не входили, то есть соответствующий бит измененных позиций одинаков. Если же сумма изменилась, то ровно одно из чисел входило в сумму, и соответствующий бит измененных позиций

разный. Иными словами, мы в точности восстанавливаем $i \oplus j$, где i и j — искомые позиции, а $\oplus$ — побитовое исключающее ИЛИ. Более того, та сумма, которая изменилась, изменилась в точности на $\pm(p_i - p_j)$.

Такой код хранит порядка $\log_2 n$ сумм, каждая из которых порядка n^2, то есть занимает $2\log_2 n$ бит. К сожалению, этого не всегда достаточно, чтобы однозначно восстановить обменянные числа. Однако можно аналогичным образом посчитать суммы не p_i, а p_i^2, тогда мы будем знать $|p_i - p_j|$ и $|p_i^2 - p_j^2|$. Зная эти две величины, можно восстановить $p_i + p_j$, а с его помощью — и сами p_i и p_j. В сумме на такой код уходит порядка $5\log_2^2 n \approx 2000$ бит.

Подзадача 7

Чтобы еще уменьшить число хранимых бит, сделаем две вещи:

1. будем считать XOR (побитовое исключающее ИЛИ) вместо сумм, сократив число бит с $2\log_2^2 n$ до $\log_2^2 n \approx 400$;
2. рядом запишем полиномиальный хеш перестановки (на самом деле два хеша, чтобы вероятность коллизии была пренебрежимо малой) — это еще плюс ≈ 60 бит.

Поскольку суммы Хэмминга позволяют нам найти $i \oplus j$, дальше просто перебрем i, найдем соответствующий ему j за одну битовую операцию, и проверим, что при их обмене хеши возвращаются в исходные запомненные значения.

Полное решение [credit to: Сергей Золотарев]

Полное решение требует отойти от идеи кодов Хэмминга и просто придумать более подходящую хеш-функцию. Заведем следующую функцию:

$$f(k) = \left(\sum_{i=1}^{n} i^k \cdot p_i\right) \bmod (4n^{k+1}).$$

Теперь посчитаем и запишем в качестве ответа $f(1)$, $f(2)$ и $f(3)$. На это уйдет $42 + 62 + 82$ бита, что, правда, требует вычислений с использованием `__int128_t` или длинной арифметики. Теперь, если нам известны эти величины и для p, и для q, достаточно посмотреть на то, как они меняются при обмене местами p_i и p_j.

При таком обмене $f(k)$ изменилось на $-i^k p_i - j^k p_j + i^k p_j + j^k p_i$, то есть на $(p_i - p_j) \cdot (j^k - i^k)$. Таким образом, нам будут известны $(p_i - p_j) \cdot (i - j)$, $(p_i - p_j) \cdot (i^2 - j^2)$ и $(p_i - p_j) \cdot (i^3 - j^3)$ по соответствующим модулям. Перебрав i, можно с большой вероятностью однозначно восстановить единственное подходящее j. За подробным описанием требуемых преобразований предлагаем вам обратиться к исходному коду полного решения в архивах олимпиады.

Задача E. DequeQL

Автор задачи: Даниил Орешников, разработчики: Даниил Орешников и Константин Бац

Подзадачи 1 и 4

Для решения подзадачи 1 можно было написать любое неэффективное решение, которое корректно выполняло описанные в условии операции. В общем случае следует заметить, что если у дека d_2 ровно i детей, и дек d_1 лежит в нем на позиции j в нумерации с 1, то на то, чтобы достать d_1 из d_2, если d_2 уже является корневым, потребуется $\min(j, i - j + 1)$ операций `pop`.

Поэтому для четвертой подзадачи достаточно было написать аккуратную симуляцию всех операций и поддерживать для каждого дека последовательность непосредственно вложенных в него деков $\mathrm{ch}(d)$, его «родителя» $\mathrm{p}(d)$ — дек, в котором он содержится, и его позицию в родителе $\mathrm{idx}(d)$. Тогда ответ на запрос `pop_complexity(`d`)` — это

$$\min(\mathrm{idx}(d), |\mathrm{ch}(d) - \mathrm{idx}(d) + 1|) + \texttt{pop\_complexity}(\mathrm{p}(d)),$$

потому что чтобы достать дек, нужно сначала достать его родителя на верхний уровень, а затем достать этот элемент из родителя. Такую формулу можно вычислить за $\mathcal{O}(n)$.

Подзадачи 2 и 3

В этих двух подзадачах гарантировалось, что в любой момент времени в каждом деке лежит непосредственно не больше 2 или 3 других деков. В первом случае понятно, что любой элемент можно достать из своего родителя за одну операцию: один `pop_front`, если элемент первый, или `pop_back`, если последний.

В таком случае `pop_complexity` от любого дека — это просто его глубина в дереве вложенности. Поддерживать глубины вершин при всех операциях достаточно несложно: любая операция изменяет глубины целого поддерева на одну и ту же величину, а значит можно было поддерживать их с помощью декартовых деревьев по неявному ключу, построенных на эйлеровых обходах. Альтернативно можно было сделать корневую декомпозицию по запросам, в каждом блоке перестраивая все активные деревья вложенности и считая глубины, после чего каждый отдельный запрос из блока учитывать отдельно.

В третьей подзадаче появляются деки, на изъятие которых из родителя требуется две операции. Такой дек среди «детей» определенного дека может быть только один — центральный из трех. В таком случае ответ для всего его поддерева увеличивается на 1 по сравнению с описанным выше ответом для случая $|\mathrm{ch}(d)| \leqslant 2$. Такие добавления и вычитания тоже можно аккуратно поддерживать с помощью ДД на эйлеровых обходах. Эйлеровы обходы в данном случае стоит хранить, выписывая каждую вершину только один раз — при входе в нее. Время работы в обоих случаях — $\mathcal{O}((n+m)\log n)$.

Подзадача 5

Эта подзадача отличается от предыдущих тем, что в ней достаточно просто просимулировать операции над деками, построив лес деревьев вложенности, после чего одним проходом `dfs` посчитать для каждой вершины ее `pop_complexity`, используя указанные выше формулы. Затем предподсчитанные значения можно использовать для ответов на запросы.

Подзадача 6

Если отсутствуют операции `pop`, то деревья вложенности только растут. Можно заметить, что в таком случае `pop_complexity` каждого дека не уменьшается, и, более того, как только какой-то дек перестает быть корневым, он больше никогда корневым не станет.

Если раскрыть рассмотренную выше сумму, можно заметить, что в процессе жизни любого дека в этой сумме меняется только последнее слагаемое, и иногда (когда корневой дек добавляется в другой), в конец добавляется новое слагаемое, равное 1. Опять же, как и было отмечено в решениях для третьей и четвертой подзадач, операцию «добавить 1» на поддереве вершины можно реализовать с помощью ДД на эйлеровых обходах. А операцию увеличения последнего слагаемого в сумме можно обрабатывать более неявно:

1. для каждого дека разобьем все время на периоды с одинаковым предком-корнем;
2. для нескольких деков с одинаковым предком-корнем все следующие периоды будут одинаковы (период заканчивается, когда предок-корень добавляется в другой дек, и тот становится новым предком-корнем);
3. будем поддерживать в начале периода текущий `pop_complexity` всех деков;
4. во время периода для ответа на запрос надо просто взять известную на начало периода сумму, после чего учесть, сколько в предка-корня было добавлено на этот момент детей с каждой стороны;

5. в конце периода надо зафиксировать и пересчитать `pop_complexity` всех деков — для этого заметим, что у каждого непосредственного ребенка текущего предка-корня «расстояние» до края родительского дека изменилось на известную величину, и можно просто сделать обновление в ДД на эйлеровом обходе для всего его поддерева.

Подзадача 7

Ну и наконец, для полного решения надо было заметить, что при каждой операции `push` или `pop` величина `pop_complexity` изменяется для всех детей дека d_2, находящихся между «серединой» (центральным по позиции ребенком) и изменяемым краем, ровно на 1.

А последовательные дети одного дека вместе со своими поддеревьями, если говорить в терминах деревьев вложенности, образуют непрерывный отрезок в эйлеровом обходе. Поэтому если хранить для каждого дека размер его поддерева и всех его детей, то на каждую операцию достаточно сделать $+1$ или -1 к отрезку в эйлеровом обходе, отвечающему за левую или правую половину детей соответствующего дека (который можно выделить, если реализовать `split` по указателю на вершину, а не по количеству вершин). Соответственно, требуется просто небольшая аккуратно реализованная модификация к описанному выше для подзадач 3 и 4 решению. Время работы остается $\mathcal{O}((n+m)\log n)$.