



Разбор задачи «Забывчивый Шустрик и пульт»

Краткое условие

Есть число A и фиксированное число M . За один шаг можно выполнить одну из операций:

$$A \leftarrow A + M, \quad A \leftarrow A - M, \quad A \leftarrow A \cdot M, \quad A \leftarrow A \bmod M.$$

Нужно найти кратчайшую последовательность операций, после которой получится число A' , удовлетворяющее

$$A' \equiv B \pmod{N}.$$

Если такой последовательности нет, нужно вывести -1 .

Главное наблюдение

Нас интересует не само число A' , а только его остаток по модулю N .

Если мы уже используем только операции

$$+, -, *,$$

то все переходы по модулю N определяются однозначно:

$$x \rightarrow (x + M) \bmod N, \quad x \rightarrow (x - M) \bmod N, \quad x \rightarrow (x \cdot M) \bmod N.$$

Значит, если забыть про операцию $\%$, задача превращается в поиск кратчайшего пути в графе из N вершин, где вершина — это остаток по модулю N .

Почему операция $\%$ сильно ограничена

Рассмотрим значение числа по модулю M .

- Операция $+$ не меняет остаток по модулю M .
- Операция $-$ не меняет остаток по модулю M .
- Операция $\%$ переводит число в диапазон $[0, M - 1]$, но его остаток по модулю M остаётся тем же.
- Операция $*$ зануляет остаток по модулю M , так как

$$A \cdot M \equiv 0 \pmod{M}.$$

Отсюда следует:

Пока не было операции $*$, остаток числа по модулю M всё время равен исходному $A \bmod M$. После первой операции $*$ остаток по модулю M становится нулём и уже никогда не меняется.

Теперь посмотрим на операцию $\%$. Она не меняет остаток по модулю M , а просто заменяет число на его наименьшего неотрицательного представителя.

Значит, операция $\%$ может быть полезна только для того, чтобы сразу перейти в одно из двух специальных состояний:

- в число $A \bmod M$, если до этого не было операции $*$;
- в число 0 , если перед этим уже была операция $*$.



Ключевая лемма о виде оптимального ответа

Лемма. В оптимальной последовательности операция $\%$ может встретиться только в одном из трёх случаев:

1. её нет совсем;
2. она стоит первой;
3. перед ней стоит ровно одна операция $*$, и эта пара имеет вид $*\%$.

Доказательство.

Операции $+$ и $-$ не меняют остаток по модулю M , а операция $\%$ лишь заменяет число на представитель того же остатка в диапазоне $[0, M - 1]$.

Поэтому если перед операцией $\%$ были только операции $+$ и $-$, то весь этот префикс никак не влияет на результат применения $\%$: после него всё равно получится число

$$A \bmod M.$$

Следовательно, такой префикс можно удалить, и путь станет короче. Значит, если операция $\%$ используется до первой $*$, то она должна стоять первой.

Теперь рассмотрим случай, когда до $\%$ уже была операция $*$. После первой $*$ число становится кратным M , то есть его остаток по модулю M равен нулю. Тогда после $\%$ обязательно получится число 0.

Если между этой первой $*$ и $\%$ были ещё какие-то операции, то они не меняют остаток по модулю M , а значит после $\%$ всё равно получится 0. Следовательно, все такие промежуточные операции можно удалить. Значит, если $\%$ используется после появления нулевого остатка по модулю M , то в оптимальном ответе она идёт сразу после первой $*$, то есть как $*\%$.

Основное решение

Теперь задача полностью сводится к трём возможным стартам:

1. начать из $A \bmod N$ — это случай, когда мы вообще не используем $\%$;
2. начать из $(A \bmod M) \bmod N$ — это случай, когда первая операция $\%$;
3. начать из 0 — это случай, когда первые две операции $*\%$.

После этого остаются только операции $+$, $-$, $*$, которые уже можно рассматривать целиком по модулю N .

Следовательно, для каждого из трёх стартов нужно найти кратчайший путь до остатка B в графе остатков по модулю N .

Граф состояний

Вершины графа:

$$0, 1, 2, \dots, N - 1.$$

Из вершины v есть до трёх переходов:

1. $v \rightarrow (v + M) \bmod N$,
2. $v \rightarrow (v - M) \bmod N$,
3. $v \rightarrow (v \cdot M) \bmod N$.



Все рёбра имеют вес 1, значит кратчайший путь ищется обычным BFS.

Разбор задачи «Грандиозное переименование»

Прежде всего, скажем, что необходимое условие существования ответа: мультимножество букв по всем строкам s_i должно совпадать с мультимножеством букв по всем строкам t_i . Если это не так — ответ -1 .

Полное решение получается из решения подзадач 2 и 3, разберём их.

Подзадача 2 (для каждого i : $\text{sorted}(s_i) = \text{sorted}(t_i)$):

Заметим, что в оптимальном решении не нужно переносить буквы между разными строками. Действительно, если буква перешла из строки i в строку j , то позднее равная ей буква должна вернуться из некоторого k в i (так как мультимножества совпадают). Но тогда вместо двух этих операций можно было бы сразу сделать перенос буквы из k в j , не увеличив суммарную стоимость. Значит, задача распадается по индексам: для каждого i независимо нужно превратить s_i в t_i операциями вида $i = j$, каждая из которых стоит 1.

Нужно максимизировать количество букв не участвующих в операциях. Заметим, что такие буквы обязательно образуют общую подпоследовательность строк s_i и t_i . Значит, буквы, которые не нужно трогать — это в точности наибольшая общая подпоследовательность (НОП) строк s_i и t_i . Минимальное число операций равно $|s_i| - |\text{НОП}|$. НОП находится стандартным динамическим программированием за $O(|s_i| \cdot |t_i|)$.

Подзадача 3 (для каждого i : s_i и t_i не имеют общих букв):

Каждая буква обязана сменить строку хотя бы один раз, при этом более одного перемещения одной буквы заведомо неоптимально. Значит относительное положение буквы внутри каждой строки не имеет значения, и задача распадается на независимые по символам алфавита: для фиксированной буквы c выписываем позиции строк, в которых c встречается в s , и позиции строк, в которых c нужна в t . Их количества совпадают (иначе ответ -1). Нужно разбить их на пары (i, j) , минимизируя $\sum(1 + |i - j|)$.

Оптимальное разбиение достигается в порядке сортировки обоих списков. Это доказывается от обратного, пусть оптимальный ответ другой, тогда есть две пары (i_1, j_1) и (i_2, j_2) такие, что $i_1 < i_2$ и $j_1 > j_2$. В этом случае можно поменять j_1 и j_2 и суммарная стоимость $(2 + |i_1 - j_1| + |i_2 - j_2|)$ не увеличится.

Полное решение:

Для каждой строки i находим $\text{НОП}(s_i, t_i)$. Буквы, вошедшие в НОП, остаются на месте. Оставшиеся буквы обрабатываем по символам алфавита независимо: сортируем списки позиций и объединяем в пары в порядке сортировки, суммируя $1 + |i - j|$. Итоговая асимптотика: $O(n \cdot |s_i| \cdot |t_i| + 26 \cdot n)$.

Разбор задачи «Расписание олимпиад»

Общая идея

Заметим, что:

1. В каждый день имеет смысл оставить **только одну** олимпиаду, дающую максимальную пользу среди всех, поставленных в этот день. Остальные олимпиады в этот день можно “спрятать” (перенести в другой день или оставить, но они не повлияют на пользу).
2. Задача сводится к выбору **не более D** олимпиад и распределению их по различным дням (по одной на день) с сохранением порядка и ограничений на даты.
3. Невыбранные олимпиады можно разместить в любые дни (не нарушая порядок), но они не дадут вклада в ответ.



Группа 1 (10 баллов)

$n, D \leq 20, b_i \leq 20, t = 1$

Решение: Переберем все 2^n подмножеств олимпиад, которые могут дать пользу. Для фиксированного подмножества размера k нужно проверить, можно ли разместить эти олимпиады в k различных днях.

Проверка: Будем размещать выбранные олимпиады жадно слева направо: очередную олимпиаду поставим в самый левый допустимый для неё день.

Сложность: $O(2^n \cdot n)$.

Группа 2 (7 баллов)

$n, D \leq 20, b_i \leq 20$

Решение: Дополняем подход из группы 1: после выбора подмножества “полезных” олимпиад нужно разместить **все** олимпиады. Перебираем олимпиады, ставим очередную в самый левый допустимый для неё день. **Сложность:** $O(2^n \cdot n)$.

Группы 3-4 (20 баллов)

$n, D \leq 800, b_i \leq 800$

Решение: Динамическое программирование: $dp[i][d]$ — максимальная польза для первых i олимпиад, если мы рассмотрели первые d дней (т.е. все олимпиады размещены в дни $\leq d$, а дни $> d$ еще не использовались).

Переход: Рассматриваем i -ю олимпиаду. Перебираем день x от $\max(d_i, d)$ до D , куда ее поставим:

$$dp[i+1][x] = \max(dp[i+1][x], dp[i][d] + b_i)$$

Если олимпиаду не учитываем в ответе, то просто переходим в состояние $i+1, d$.

Сложность: $O(n \cdot D^2)$.

Группы 5-6 (9 балла)

$n, D \leq 2500, b_i \leq 2500$

Решение (для групп 5–6): Более эффективное ДП с двумя типами переходов:

Состояние $dp[i][d]$ — максимальная польза для первых i олимпиад, если мы рассмотрели первые d дней, **и день d уже закрыт** (т.е. мы больше не можем ставить учтённые олимпиады в день d).

Переходы:

- Олимпиада i ставится в день d (и не учитывается): $dp[i+1][d] = \max(dp[i+1][d], dp[i][d])$
- Ставим i -ю олимпиаду в день $d+1$: $dp[i+1][d+1] = \max(dp[i+1][d+1], dp[i][d] + b_i)$ (при условии $d+1 \geq d_i$)
- Пропускаем день d (олимпиада i ставится позже): $dp[i][d+1] = \max(dp[i][d+1], dp[i][d])$

Сложность: $O(n \cdot D)$.

Группа 7 (5 баллов)

Все b_i равны

Решение: Так как все пользы одинаковы, задача сводится к максимизации **количества дней**, в которых есть хотя бы одна олимпиада. Жадность: идем по дням от 1 до D . Для каждого дня пытаемся найти самую левую из еще не размещенных олимпиад, которую можно поставить в этот день (т.е. $d_i \leq$ текущий день). Если находим — ставим, иначе день остается пустым. Это максимизирует количество занятых дней.

Достаточно рассмотреть последние n дней.

Сложность: $O(n + D)$.



Группа 8 (5 баллов)

Все d_i различны

Решение: Оставим каждую олимпиаду на ее исходной дате. Тогда все дни заняты разными олимпиадами, и суммарная польза = $\sum b_i$.

Сложность: $O(n)$.

Группа 9 (5 баллов)

Все $d_i = 1$, $t = 1$

Решение: Все олимпиады изначально в день 1. Мы можем сдвинуть их на любые дни ≥ 1 . Нам нужно выбрать $\min(n, D)$ олимпиад (по одной на день) с максимальной суммарной пользой. Берем $\min(n, D)$ олимпиад с максимальной пользой (при равенстве пользы — любые). Расставляем их в порядке номеров по дням $1, 2, \dots$

Сложность: $O(n \log n)$.

Группа 10 (5 баллов)

Все d_i равны (общая дата d_0)

Решение: Доступные дни: от d_0 до D , всего $D - d_0 + 1$ дней. В каждый такой день можно поставить не более одной олимпиады. Выбираем $\min(n, D - d_0 + 1)$ олимпиад с максимальной пользой и расставляем в порядке номеров.

Сложность: $O(n \log n)$.

Группы 11-13 (16 баллов)

Без дополнительных ограничений

Полное решение (жадность с конца):

1. Будем идти по дням от D до 1.
2. Поддерживаем **активное множество** олимпиад, которые мы **планируем учесть** в ответе, и структуру данных для быстрого извлечения минимальной пользы (например, `min-heap`).
3. Для текущего дня j добавляем в активное множество все олимпиады с $d_i \leq j$ (их можно перенести в день j или позже).
4. Если в активном множестве больше элементов, чем осталось дней (включая текущий), удаляем из него олимпиаду с **наименьшей** пользой (она не войдет в ответ).
5. После прохода всех дней в активном множестве останутся олимпиады, которые дадут максимальную пользу.

Почему это работает: На каждом шаге мы гарантируем, что для оставшихся дней (с конца) есть достаточно места, чтобы разместить все выбранные олимпиады в порядке номеров. Удаление наименее полезной олимпиады оптимально, так как если нужно освободить место, лучше пожертвовать олимпиадой с наименьшей пользой.

Сложность: $O((n + D) \log n)$.

Примечание: В данном решении мы сначала оставили только ограничение об относительном порядке и проигнорировали второе ограничение (ограничение на неуменьшение дат), а затем вернули его (стали игнорировать *лишние* олимпиады с минимальной полезностью). Возможно и обратное — оставить только ограничение на неуменьшение дат. Тогда жадное решение (идти слева направо и ставить олимпиаду с максимальной полезностью) даст нам только набор олимпиад, но не сохранит относительный порядок. Останется лишь посортировать выбранные олимпиады по номеру и расставить их в последние k дней.

Разбор задачи «Башня из коробок»



Одно из возможных решений

Одно из самых простых решений — новый предмет размещать всегда наверху стека, а при извлечении предмета — собирать все блокирующие его извлечение предметы в буфер, извлекать предмет и возвращать предметы из буфера в стек в отсортированном порядке — чем раньше извлекается предмет, тем выше в стеке мы его расположим. Такое решение не всегда будет оптимальным, например, из состояния стека $[\dots, 2, 1, 3, 4]$ (здесь самый правый элемент — наверху стека) перейдём в состояние $[\dots, 2, 4, 3]$ при извлечении предмета с номером 1. Затем при извлечении предмета с номером 2 предметы 3, 4 будут повторно помещены в буфер. Видно, что если изначально вместе с предметом 1 достать ещё один, следующий предмет — 2, суммарное число операций уменьшится. А именно от состояния стека $[\dots, 2, 1, 3, 4]$ мы перейдём к состоянию $[\dots, 4, 3, 2]$, после чего извлечём каждый из предметов за одно действие сверху стека. В этом случае предметы 2, 3, 4 по одному разу окажутся в буфере и число действий сократится.

Способ 1 (12 действий)

Шаг	Стек	Буфер	Действия
1	$[\dots, 2, 1, 3, 4]$	$\emptyset$	Помещаем в буфер 4, 3
2	$[\dots, 2, 1]$	$\{3, 4\}$	Извлекаем 1
3	$[\dots, 2]$	$\{3, 4\}$	Размещаем из буфера 4, 3
4	$[\dots, 2, 4, 3]$	$\emptyset$	Помещаем в буфер 3, 4
5	$[\dots, 2]$	$\{3, 4\}$	Извлекаем 2
6	$[\dots]$	$\{3, 4\}$	Размещаем из буфера 4, 3
7	$[\dots, 4, 3]$	$\emptyset$	Извлекаем 3, 4
8	$[\dots]$	$\emptyset$	

Способ 2 (10 действий)

Шаг	Стек	Буфер	Действия
1	$[\dots, 2, 1, 3, 4]$	$\emptyset$	Помещаем в буфер 4, 3
2	$[\dots, 2, 1]$	$\{3, 4\}$	Извлекаем 1
3	$[\dots, 2]$	$\{3, 4\}$	Помещаем в буфер 2
4	$[\dots]$	$\{2, 3, 4\}$	Размещаем из буфера 4, 3, 2
5	$[\dots, 4, 3, 2]$	$\emptyset$	Извлекаем по очереди 2, 3, 4
6	$[\dots]$	$\emptyset$	

Однако рассмотрим оба способа в случае, когда после извлечения предмета 1 прибывают для размещения предметы 5, 6.

Случай 2. Способ 1 (20 действий)

Шаг	Стек	Буфер	Действия
1	$[\dots, 2, 1, 3, 4]$	$\emptyset$	Помещаем в буфер 4, 3
2	$[\dots, 2, 1]$	$\{3, 4\}$	Извлекаем 1
3	$[\dots, 2]$	$\{3, 4\}$	Размещаем из буфера 4, 3
4	$[\dots, 2, 4, 3]$	$\emptyset$	Размещаем 5, 6
5	$[\dots, 2, 4, 3, 5, 6]$	$\emptyset$	Помещаем в буфер 6, 5, 3, 4
6	$[\dots, 2]$	$\{3, 4, 5, 6\}$	Извлекаем 2
7	$[\dots]$	$\{3, 4, 5, 6\}$	Размещаем из буфера 6, 5, 4, 3
8	$[\dots, 6, 5, 4, 3]$	$\emptyset$	Извлекаем по очереди 3, 4, 5, 6
9	$[\dots]$	$\emptyset$	



Случай 2. Способ 2 (22 действия)

Шаг	Стек	Буфер	Действия
1	[..., 2, 1, 3, 4]	$\emptyset$	Помещаем в буфер 4, 3
2	[..., 2, 1]	{3, 4}	Извлекаем 1
3	[..., 2]	{3, 4}	Помещаем в буфер 2
4	[...]	{2, 3, 4}	Размещаем из буфера 4, 3, 2
5	[..., 4, 3, 2]	$\emptyset$	Размещаем 5, 6
6	[..., 4, 3, 2, 5, 6]	$\emptyset$	Помещаем в буфер 6, 5
7	[..., 4, 3, 2]	{5, 6}	Извлекаем 2
8	[..., 4, 3]	{5, 6}	Помещаем в буфер 4, 3
9	[...]	{3, 4, 5, 6}	Размещаем из буфера 6, 5, 4, 3
10	[..., 6, 5, 4, 3]	$\emptyset$	Извлекаем по очереди 3, 4, 5, 6

Значит, наши текущие действия могут казаться оптимальными, но с учётом предметов, которые придут в будущем, оптимальным набором действий может оказаться уже другой набор.

Предложенные решения

Здесь и далее T обозначает максимальное число элементов, которое может одновременно находиться в стеке.

Решение за $O(N\sqrt{T})$ действий

Разобьём события на блоки по $\sqrt{T}$ событий. Перед началом блока за не более чем T действий сможем поместить минимальные $\sqrt{T}$ элементов на верхушку стека. После этого после i -го события в блоке будем поддерживать состояние таким, что хотя бы $\sqrt{T} - i$ минимальных элементов находятся в верхних $\sqrt{T}$ элементах. Поддерживать такое состояние несложно — действительно, после очередного извлечения элемента число минимальных в верхних $\sqrt{T}$ уменьшилось не более чем на один, а при размещении нового элемента один минимальный мог быть сдвинут вглубь стека. При этом для получения очередного элемента достаточно каждый раз $\sqrt{T}$ действий — ведь как минимум $\sqrt{T} - i > 0$ минимальных элементов находится у верхней границы стека на i -ом шаге в блоке. Тогда один раз в $\sqrt{T}$ событий такое решение потратит $O(T)$ действий, и на каждое извлечение не более чем $\sqrt{T}$ действий. Суммарное число действий для данного решения не превосходит

$$O(N\sqrt{T}) = N\sqrt{T} + \frac{N}{\sqrt{T}}T.$$

Оценка на число действий

Рассмотрим пример — пусть N предметов приходит и уходит в порядке очереди, при этом размер очереди равен T . А именно в момент времени $2i$ появляется элемент со значением $t_i = i + 2T + 1$. Тогда с момента времени $2T$ предметы по очереди размещаются и извлекаются из стека.

Обозначим число элементов, снятых со стека в момент i как a_i . Заметим, что число моментов времени, когда $a_i \geq q$ для любого целого $q < T$ не меньше чем $(N - T)/q$. Докажем это.

Допустим q шагов подряд значение a_i не превосходит q (за шаг считаем извлечение предмета и затем размещение нового предмета). Тогда элемент x , который необходимо будет извлечь в момент сразу после этих q шагов будет хотя бы на глубине q в стеке по той причине, что в верхних q элементах стека с каждым новым размещением появляется один элемент, больший x (причём на каждом шаге один элемент из верхних q заменяется на новый, и никакой элемент не мог сначала заменить, а затем заместиться, следовательно заменились все элементы и не осталось элементов $\leq x$). Значит на следующем шаге, чтобы получить доступ к элементу x и извлечь его необходимо потратить хотя бы $q + 1$ действие.



Рассмотрим значения $q = 2^m$ для всех $2^m < T$. Тогда для каждого такого значения q будет хотя бы $N/(q+1)$ моментов времени таких, что $a_i \geq q$. При этом один момент времени будет учтён несколько раз (для разных значений q). Тогда каждому значению m приведём в соответствие $(N/2^m - N/2^{m+1})$ моментов времени, начиная с $m = \log_2 T$ и заканчивая $m = 1$, каждый раз исключая моменты времени уже занятые до этого. Получим что суммарно действий было совершено порядка

$$N \log T \approx \sum_{m=1}^{\log T} \left(\frac{N}{2^{m+1}} \right) \cdot 2^m.$$

При этом приведённая асимптотическая оценка совпадает с верхней оценкой на число действий — далее приведём решение, которое для произвольных значений времени извлечения выполнит не более $O(N \log T)$ действий, где T — наибольшее число элементов, которое хранится в стеке одновременно (если этот параметр известен заранее).

Решение за $O(N \log T)$ действий

Будем считать число событий (появление нового элемента для размещения и извлечение очередного элемента), произошедших с начала работы алгоритма. Тогда после события j для наибольшей степени двойки m , такой что $2^m \mid j$ (то есть j делится нацело на 2^m) достанем из стека $\min(2 \cdot 2^m, |S|)$ элементов и вернём обратно в порядке убывания времени извлечения.

Лемма: Перед действием в момент j следующие $\min(2^m, |S|)$ по времени извлечения предметов (не учитывая ещё не поступившие в стек) находятся среди верхних $2 \cdot 2^m$ элементов стека.

Доказательство: Докажем по индукции по j . База индукции: для $j = 0$ в стеке находится нулевое количество элементов, для них утверждение леммы выполняется.

Переход. Заметим что после обработки события $j - 2^m$ утверждение выполняется (так как $j - 2^m < j$) по предположению индукции. $2^{m+1} \mid j - 2^m$, а значит $\min(2 \cdot 2^{m+1}, |S|)$ минимальных по времени извлечения элементов упорядочены и находятся наверху стека после обработки события $j - 2^m$.

Пусть среди событий $[j - 2^m, j)$ было r извлечений и $2^m - r$ размещений. Каждое извлечение забирает минимальный элемент из верхних $2 \cdot 2^m$, каждое размещение сдвигает самый поздний из верхних элементов вглубь стека. Тогда перед обработкой события j в верхних $2 \cdot 2^m$ элементах осталось хотя бы

$$2 \cdot 2^m - 2^m = 2^m$$

минимальных элементов, минимальных по времени извлечения, что равно требуемому количеству $\min(2^m, |S|)$.

Упражнение для читателя:

Придумайте решение за $O(N^{4/3})$.

Разбор задачи «Чэн Ши и коллекции бамбуковых надписей»

Краткое условие

Поддерживается динамическое мультимножество строк. После каждой операции добавления или удаления строки нужно найти минимальное число действий, необходимое, чтобы сделать все строки одинаковыми.

Разрешённые действия над одной строкой:

- удалить последний символ;
- дописать любой символ в конец.

Строка, к которой мы приводим все остальные, может быть любой, в том числе пустой.



Наблюдение 1: переход к дереву-префиксов

Построим бор по всем строкам, которые вообще встречаются во входных запросах. Корень соответствует пустой строке, каждая вершина — некоторому префиксу, а глубина вершины равна длине соответствующей строки.

Каждая строка из мультимножества теперь отождествляется с вершиной бора.

Если мы хотим превратить строку s в строку t , то:

- нужно удалить из s суффикс после их наибольшего общего префикса;
- затем дописать недостающий суффикс строки t .

Поэтому число действий равно

$$|s| + |t| - 2 \cdot |LCP(s, t)|.$$

Но именно так выглядит расстояние между двумя вершинами в боре. Значит, стоимость приведения строки s к строке t равна расстоянию между соответствующими вершинами дерева.

Итак, задача сводится к следующей:

Есть дерево (бор), в некоторых вершинах лежат активные фишки с кратностями. После каждого добавления/удаления фишки нужно найти вершину v , минимизирующую сумму расстояний до всех активных фишек.

Почему достаточно рассматривать только вершины бора

В условии разрешено приводить все строки к любой строке, даже если её нет в мультимножестве. Но достаточно рассматривать только вершины бора.

Действительно, если целевая строка не является префиксом ни одной строки, встречающейся во входе, то соответствующей вершины в боре нет. Тогда последняя буква этой строки ведёт в поддерево, в котором нет ни одной активной строки. Если сдвинуть цель на один символ вверх, расстояние до каждой активной строки не увеличится, а на самом деле уменьшится на 1. Значит, оптимальная цель всегда является вершиной бора, построенного по всем строкам запросов.

Наблюдение 2: оптимальная вершина — медиана дерева

Пусть в дереве выбрана вершина v , а суммарная стоимость равна

$$F(v) = \sum_{u \text{ активна}} \text{dist}(u, v),$$

где строки учитываются с кратностями.

Рассмотрим переход из вершины v в её соседа to . Пусть в компоненте, которая содержит to после удаления ребра (v, to) , лежит x активных строк, а всего активных строк M .

Тогда:

- для этих x строк расстояние уменьшится на 1;
- для остальных $M - x$ строк расстояние увеличится на 1.

Следовательно,

$$F(to) - F(v) = (M - x) - x = M - 2x.$$

Отсюда немедленно следует критерий оптимальности.

Лемма. Вершина v минимизирует сумму расстояний до всех активных строк тогда и только тогда, когда после удаления v каждая компонента содержит не более половины всех активных строк.



Доказательство. Если в некоторую компоненту уходит больше половины активных строк, то переход в эту сторону уменьшает ответ, так как $M - 2x < 0$. Если же во всех компонентах не более половины, то переход в любую сторону ответ не улучшает.

Это обычная взвешенная медиана дерева.

Как найти медиану в корневом дереве

Зафиксируем корень бора в пустой строке.

Пусть всего активных строк M . Тогда медиана — это вершина, в которой:

- ни одно поддерево сына не содержит более $M/2$ активных строк;
- при этом выше подниматься уже нельзя, иначе мы попадём в компоненту “всё остальное”, содержащую больше половины.

Эквивалентно: если идти от корня вниз, то пока существует сын, чьё поддерево содержит строго больше половины всех активных строк, нужно идти в него. Остановимся мы ровно в медиане.

То есть медиана — это самая глубокая вершина, чьё поддерево содержит больше половины всех активных строк.

Как находить такую вершину быстро

Будем хранить активные строки в вершинах trie с кратностями.

Сделаем эйлеров обход по бору. Тогда каждому поддереву соответствует отрезок:

$$subtree(v) \leftrightarrow [tin[v], tout[v]].$$

Если хранить в дереве фенвика или дереве отрезков количество активных строк в точках $tin[v]$, то число активных строк в поддереве вершины v можно получать как сумму на отрезке эйлерового обхода.

Теперь хотим найти медиану. Для этого удобно сделать два шага.

Шаг 1. Найдём в порядке эйлерова обхода первую активную строку, после которой префиксная сумма становится больше

$$\left\lfloor \frac{M}{2} \right\rfloor.$$

Обозначим соответствующую вершину через x .

Шаг 2. Медиана лежит на пути от корня к x . Действительно, поддерево медианы содержит больше половины всех активных строк, значит момент “переваливания через половину” в эйлеровом обходе обязательно произойдёт внутри этого поддерева.

Теперь среди предков вершины x нужно найти самую глубокую вершину v , для которой

$$cnt(subtree(v)) > \frac{M}{2}.$$

Именно она и будет медианой.

Это можно сделать бинарным подъёмом по предкам: мы умеем быстро проверять для любого предка размер его поддерева, значит находим нужного предка за $O(\log^2 N)$ или быстрее.



Как быстро считать сумму расстояний до всех активных строк

После того как медиана найдена, нужно посчитать

$$\sum_{u \text{ активна}} dist(u, med).$$

Для этого используем стандартную динамику на центроидном дереве.

Что хранится в центроидном дереве

Построим центроидное дерево на боре.

Для каждой вершины-центроида c будем хранить:

- $cnt[c]$ — сколько активных строк учтено в структурах этого центроида;
- $sum[c]$ — сумма расстояний от всех активных строк до c .

Кроме того, для каждого центроида x (кроме корня центроидного дерева) храним информацию о той части, которая соответствует его компоненте относительно родителя по центроидному дереву:

- $subcnt[x]$ — сколько активных строк лежит в этой части;
- $subsum[x]$ — сумма расстояний от этих строк до родителя x в центроидном дереве.

Обновление

При добавлении или удалении активной строки в вершине v поднимаемся по всем центроидам на пути от v к корню центроидного дерева.

Для каждого такого центроида c :

- меняем $cnt[c]$;
- меняем $sum[c]$ на $\pm dist(v, c)$.

А для центроидного сына, через которого мы пришли вверх, обновляем также $subcnt$ и $subsum$. Число таких центроидов равно $O(\log N)$.

Запрос

Для произвольной вершины v сумма расстояний до всех активных строк считается так же стандартно.

Поднимаемся по пути v к корню центроидного разложения. Для каждого центроида c :

- прибавляем

$$sum[c] + cnt[c] \cdot dist(v, c),$$

что соответствует всем активным строкам, учтённым в c ;

- но часть строк, лежащих в том же центроидном поддереве, что и v , посчитана лишний раз, поэтому её нужно вычесть с помощью $subcnt$ и $subsum$.

В итоге за $O(\log N)$ верно вычисляется сумма расстояний от v до всех активных строк.

Разбор задачи «Нимсапёр!»

Сразу приведём полное решение. Задача сводится к следующей формулировке: дано дерево, все вершины которого изначально белые. Рассматривается процесс, в котором некоторые вершины последовательно перекрашиваются в чёрный цвет. После каждого шага в каждой белой вершине записано число её чёрных соседей. Требуется определить максимальное число вершин, которое можно



покрасить в некотором порядке так, чтобы после каждого из этих шагов XOR всех значений в белых вершинах был равен нулю. Если это максимальное число равно k , то ответ задачи равен $k + 1$.

Опишем изменение XOR после покраски одной вершины. Пусть в некоторый момент красится белая вершина v . Через t обозначим число её чёрных соседей в этот момент. До покраски значение t входило в общий XOR, поскольку вершина v была белой. После покраски вершина v перестаёт учитываться. Для каждого белого соседа u вершины v значение в вершине u увеличивается на единицу, то есть изменяется с a_u на $a_u + 1$. Следовательно, общий XOR изменяется на величину

$$t \oplus \bigoplus_{u \text{ — белый сосед } v} (a_u \oplus (a_u + 1)).$$

Следовательно, если до рассматриваемого шага XOR был равен нулю, то после него он останется равным нулю тогда и только тогда, когда величина изменения XOR равна нулю.

Введём обозначение

$$\Delta(x) = x \oplus (x + 1).$$

Тогда условие корректности покраски вершины v принимает вид

$$t = \bigoplus_{u \text{ — белый сосед } v} \Delta(a_u).$$

Рассмотрим значения функции $\Delta(x)$. Для любого x число $\Delta(x)$ имеет вид $2^k - 1$. Это следует из двоичного представления: при переходе от x к $x + 1$ меняется конечный суффикс битов, поэтому в XOR возникает последовательность подряд идущих единиц. Например,

$$0 \oplus 1 = 1, \quad 1 \oplus 2 = 3, \quad 2 \oplus 3 = 1, \quad 3 \oplus 4 = 7.$$

Следовательно, если у фиксированной белой вершины число чёрных соседей последовательно увеличивается с 0 до 1, затем с 1 до 2, затем с 2 до 3 и так далее, то возникающие метки образуют последовательность

$$1, 3, 1, 7, 1, 3, 1, 15, 1, 3, 1, 7, 1, 3, 1, 31, \dots$$

Таким образом, каждый раз, когда красится очередной сосед фиксированной вершины, в условии сохранения нулевого XOR появляется одна из указанных меток.

Для фиксированной вершины эта последовательность определяется исключительно числом уже покрашенных соседей. Если у вершины уже покрашено k соседей, то множество использованных меток совпадает с первыми k членами указанной последовательности. Поэтому существенной является только длина соответствующего префикса.

Назовём состояние вершины префиксным, если оно соответствует первым k элементам последовательности

$$1, 3, 1, 7, 1, 3, 1, 15, \dots$$

для некоторого k . В реализации такое состояние кодируется количеством уже использованных меток каждого типа. При ограничении $n \leq 369$ достаточно учитывать только метки

$$1, 3, 7, 15, 31.$$

Докажем это. Пусть $s[k]$ обозначает минимальное число вершин в дереве, необходимое для существования корректной конфигурации, в которой некоторая белая вершина имеет ровно k чёрных соседей. Для перехода от $k - 1$ к k требуется покрасить ещё одного её соседа. При этом возникает метка

$$(k - 1) \oplus k = 2^x - 1$$



Чтобы после такого шага общий XOR снова стал равен нулю, внутри поддерева нового соседа должна существовать конфигурация, компенсирующая именно эту метку. Минимальное число вершин, необходимое для этого, не меньше $c[2^x - 1]$. Следовательно,

$$c[k] \geq c[k - 1] + c[2^x - 1].$$

Из базовых значений $c[1] = 1$, $c[2] = 3$, $c[3] = 7$ и этой рекуррентной оценки следует, что уже $c[63] > 369$. Следовательно, метка 63 и все большие метки не могут появиться ни в одной допустимой конфигурации при заданных ограничениях. Поэтому достаточно рассматривать только метки 1, 3, 7, 15, 31.

Задача решается динамикой по дереву. Корень выбирается произвольно. Для каждой вершины v рассматривается её поддерево. Относительно родителя существуют два случая: вершина v покрашена раньше родителя либо после него. Случай, когда вершина v не покрашивается в рассматриваемом хорошем префиксе, включается во второй случай.

Состояние *before_parent*[l] означает, что вершина v покрашена раньше родителя, а в момент её покраски увеличение числа чёрных соседей у родителя создаёт метку l . Иными словами, из поддерева вершины v вверх передаётся метка l . Значение этого состояния равно максимальному числу покрашенных вершин в поддереве v , достижимому при указанном условии.

Состояние *after_parent*[l] означает, что родитель покрашен раньше вершины v . При этом либо вершина v будет покрашена позднее и тогда в момент своей покраски передаст наверх метку l , либо вершина v не будет покрашена в рассматриваемом хорошем префиксе. Частный случай $l = 0$ соответствует отсутствию вклада наверх. Значение этого состояния также равно максимальному числу покрашенных вершин в поддереве v .

Рассмотрим сначала случай, когда вершина v остаётся белой. Тогда для каждого её ребёнка существуют два допустимых варианта. В первом варианте ребёнок к этому моменту также не покрашен; тогда соответствующее дочернее поддерево должно находиться в состоянии *after_parent*[0]. Во втором варианте ребёнок покрашен раньше вершины v ; тогда он использует некоторую метку l из префиксной последовательности вершины v , а его поддерево должно находиться в состоянии *before_parent*[l]. После последовательной обработки детей формируется некоторое префиксное состояние вершины v , то есть фиксируется, какие первые метки последовательности 1, 3, 1, 7, ... уже реализованы её ранее покрашенными детьми. Для каждого такого состояния хранится максимум числа покрашенных вершин в уже обработанной части поддерева.

Рассмотрим случай, когда вершина v сама покрашена. Тогда её дети разбиваются на две группы. К первой группе относятся дети, покрашенные раньше вершины v ; они образуют некоторое префиксное состояние, обозначим его через s . Ко второй группе относятся дети, которые в момент покраски вершины v ещё белые; они либо будут покрашены позже, либо не будут покрашены вовсе.

Пусть префиксное состояние s соответствует множеству детей вершины v , которые уже покрашены к моменту покраски v . Тогда число таких детей однозначно определяется состоянием s . Обозначим его через *total*(s). Поскольку к этому моменту родитель и остальные дети вершины v ещё не покрашены, число чёрных соседей вершины v в момент её покраски равно именно *total*(s).

Рассмотрим теперь остальных детей вершины v , то есть тех, которые в момент её покраски ещё остаются белыми. Каждый такой ребёнок даёт вклад в условие корректности покраски v через соответствующую метку. XOR всех этих вкладов обозначим через x .

Тогда условие корректности покраски вершины v формулируется следующим образом. Если родитель вершины v в этот момент отсутствует или не даёт дополнительного вклада, то суммарный вклад от ещё белых детей должен совпадать с числом уже покрашенных детей, то есть должно выполняться равенство

$$x = \text{total}(s).$$

Если же вершина v красится раньше родителя и при этом наверх передаётся метка l , то эта метка также участвует в балансе, и условие принимает вид



$$x = l \oplus total(s).$$

Следовательно, для уже покрашенной вершины v достаточно хранить два параметра: префиксное состояние s , задающее детей, покрашенных раньше вершины v , и значение x , равное XOR вкладов детей, остающихся белыми в момент её покраски. При обработке очередного ребёнка возможны два случая. Если этот ребёнок покрашен раньше вершины v , то его поддереву должно находиться в состоянии $before_parent[l]$, а метка l добавляется в префиксное состояние. Если этот ребёнок к моменту покраски вершины v ещё белый, то его поддерево должно находиться в состоянии $after_parent[l]$, а метка l XOR-ится в величину x . После обработки всех детей остаётся проверить одно из указанных выше равенств; если оно выполнено, то покраска вершины v допустима.

Теперь опишем переходы.

Для вычисления $after_parent[0]$ рассматриваются все допустимые префиксные состояния детей в случае, когда вершина v остаётся белой. Среди них выбирается максимум по числу покрашенных вершин.

Для вычисления $after_parent[l]$ при $l \neq 0$ используется следующее условие. Метка l впоследствии должна быть передана от вершины v к родителю, следовательно, она должна входить в допустимый префикс последовательности вершины v . Эта метка резервируется для связи с родителем, а остальные метки префикса формируются детьми. После этого рассматриваются два случая: вершина v остаётся белой либо вершина v покрашивается; во втором случае дополнительно проверяется условие баланса между уже покрашенными и ещё белыми детьми.

Для вычисления $before_parent[l]$ вершина v обязана быть покрашена раньше родителя. Если дети, покрашенные раньше неё, образуют префиксное состояние s , то у вершины v уже имеется $total(s)$ чёрных соседей снизу, а вклад l соответствует родителю. Следовательно, XOR вкладов от детей, не покрашенных к моменту покраски вершины v , должен быть равен

$$l \oplus total(s).$$

Среди всех состояний s , для которых это условие выполнимо, выбирается максимум.

Корректность динамики доказывается индукцией по поддереву. Для листа утверждение очевидно, поскольку детей у него нет и все состояния вычисляются непосредственно. Предположим, что для всех детей вершины v значения состояний $before_parent$ и $after_parent$ вычислены корректно. Рассмотрим допустимую конфигурацию хорошего префикса внутри поддерева вершины v . Для каждого ребёнка относительно вершины v возможен ровно один из трёх вариантов: ребёнок покрашен раньше вершины v , позже неё или не покрашен вовсе. Следовательно, поведение каждого дочернего поддерева полностью задаётся одним из состояний $before_parent[l]$ или $after_parent[l]$. При объединении детей на вершине v существенными являются только две характеристики: префиксное состояние, образованное детьми, покрашенными раньше вершины v , и XOR меток от детей, остающихся белыми в момент её покраски. Дополнительная информация о внутреннем устройстве дочерних поддеревьев для проверки корректности не требуется. Следовательно, переходы динамики перебирают все допустимые конфигурации без потерь. Поскольку в каждом состоянии выбирается максимум по числу покрашенных вершин, получаются корректные оптимальные значения состояний для вершины v . По индукции утверждение верно для всех вершин дерева, в том числе для корня.