Задача A. Алфавитно-цифровое кодирование

Обозначим за ℓ число, равное $\lceil n/x \rceil$.

Очевидно, что строка с контрольным кодом n не может состоять из более чем ℓ символов (иначе её контрольный код будет больше n, исходя из выбора ℓ). Так что если $k > \ell$, то ответ равен -1.

Рассмотрим возможные значения разности $d = n - \ell \cdot x$ для строки длиной ℓ. В случае, если все буквы равны ‘a’, то $d = 0$, если все буквы равны ‘z‘, то $d = 25 \cdot \ell$. Остальные значения лежат между этими двумя. Из этого следует, в частности, что при $d > 25 \cdot \ell$ строк длины ℓ с контрольным кодом n не существует.

Более того, в этом случае строк с требуемым контрольным кодом не существует вообще: про строки длины, большей ℓ, уже было сказано выше. Если мы берём строку длины ℓ', меньшей ℓ, то $d' = n - \ell' > n - \ell > d$, и $25 \cdot \ell' < 25 \cdot \ell < d < d'$, то есть строк меньшей длины с контрольным кодом n также не существует.

В случае, если разность не превосходит $25 \cdot \ell$, а k не превосходит ℓ, требуемые строки существуют. Чтобы получить лексикографически наименьшую из них, заметим, что если есть возможность добавить ещё одну букву ‘a‘ вначале, её надо добавлять (если мы выбираем «не добавлять», то строка не будет лексикографически наименьшей: строка, в которой на этом месте стоит ‘a‘, будет меньше). Возможность добавить i букв ‘a‘ вначале есть до тех пор, пока $25 \cdot (n - i) \geqslant d$. Находим наибольшее такое i, далее находим код первой буквы после (возможно, пустого) блока букв ‘a‘ как остаток от деления d на 25. Остальные буквы, если они есть, имеют код 25, то есть являются буквами ‘z‘.

Отметим, что при вычислениях лучше использовать тип `long long`, иначе есть риск переполнения.

Задача B. Битовые операции возвращаются

Для решения первой подзадачи достаточно было предпросчитать за линейное время до всех запросов массив префиксных максимумов, после чего отвечать на запросы за $O(1)$.

Во второй подзадаче можно было написать полный перебор подпоследовательностей и проксорить все их максимумы.

В третьей подзадаче можно было реализовать $F(B, k)$, где B – подпоследовательность исходного массива, в рекурсивном виде. Сложность такого решения равна $O((k+1)^n)$, так как каждый элемент может быть в одном из $k + 1$ состояний – не принадлежать ни одной подпоследовательности; принадлежать одной и не принадлежать последующим; принадлежать двум и так далее. Здесь имеются в виду последовательности, которые получаются при вычислении функции F.

Сформулируем главную идею, необходимую для решения последующих двух подзадач.

Давайте отсортируем массив A и попробуем вычислить $F(A, 1)$. Тогда первый элемент может быть максимумом в одном множестве, второй – в двух, третий – в четырех; i-й – в 2^{i-1} множествах. Заметим, что мы ксорим каждый максимум столько раз, во сколько множеств он входит. Но каждая степень двойки, кроме нулевой, четна. Используя свойство ксора $x \oplus x = 0$, получим, что все элементы, кроме первого, сократятся, и результатом будет только минимальный элемент массива! То есть, $F(A, 1) = min(A)$. Но совершенно аналогично можно показать, что $F(A, 2) = max(A)$ и т. д.

Таким образом, для полного решения предпросчитаем префиксные максимумы и минимумы. Тогда, если k нечетно, выводим префиксный минимум, иначе максимум.

В предпоследней подзадаче минимум и максимум на префиксе можно находить за линейное время, проходясь по префиксу циклом.

Сложность решения – $O(n + q)$.

Задача C. Восстановите последовательность

Если y не делится на x, то таких последовательностей нет (НОК множества чисел делится на каждое из чисел, то есть делится на все его делители, включая и наибольший общий делитель), и мы выводим 0.

Так как НОД всех элементов последовательности равен x, это значит, что каждый элемент последовательности обязан делиться на x. Разделив все элементы последовательности на x, получим соответствие последовательности из условия задачи и последовательности, для которой НОД всех

элементов равен 1, а НОК равен y/x. Таким образом, достаточно найти самую длинную строго возрастающую последовательность элементов, у которой НОД равен 1, а НОК задан. Покажем, что эта последовательность — отсортированное множество всех различных делителей числа y/x.

Так как НОК всех элементов равен y/x, то НОК должен делиться на все элементы последовательности (по определению наименьшего общего кратного). Так как последовательность является строго возрастающей, то все её элементы различны, то есть все такие последовательности есть подмножество множества всех различных делителей числа y/x, отсортированное по возрастанию. Тем самым наибольшая такая последовательность строится из множества различных делителей числа y/x. Длина такой последовательности равно произведению чисел $(a_i + 1)$, где a_i — показатель степени, с которым простое число p_i входит в разложение числа y/x. Поэтому для вычисления длины требуется факторизовать число y/x.

Учитывая, что для 10^5 запросов факторизация «в лоб» может не сработать, для решения задачи на полный балл требуется провести некоторые оптимизации (например, построить вектор простых чисел до 10^4. Авторское решение строит решето Эратосфена для чисел до 10^8 включительно, в котором элемент имеет тип `short` (двухбайтовый целочисленный тип), и равен 0 для простых чисел и наименьшему простому делителю для составных. С помощью этой таблицы ответ на запрос работает за логарифмическую сложность, и решение укладывается в ограничения по времени даже для T порядка 10^6.

Задача D. Главное — любить задачи

Во первых, переформулируем задачу. Необходимо посчитать количество непересекающихся пар отрезков таких, что сумма их элементов равна ксору (исключающему битовому или) их элементов.

Для решения первой подгруппы можно написать любой полный перебор всех пар отрезков.

Решение второй подгруппы подразумевает полный перебор пар отрезков, с проверкой каждой пары за $\mathcal{O}(1)$ времени. Можно использовать префиксные суммы и префиксные ксоры, или же просто во время прохода циклом по второй границе отрезка поддерживать сумму и ксор. Итого $\mathcal{O}(n^4)$, т.к. каждая граница отрезка имеет не более чем $n + 1$ вариант значения.

Для дальнейших рассуждений заметим, что сумма на отрезке всегда больше или равна ксору, а значит мы можем рассмотреть каждый отрезок отдельно, и посчитать количество таких пар отрезков, что сумма на каждом отдельном отрезке равна ксору. Назовём такие отрезки валидными.

В третьей подгруппе можно перебрать все одиночные отрезки, и для каждого валидного отрезка $[l, r]$ запомнить, что в границе r заканчивается на один отрезок больше. Потом нужно просуммировать по префиксам количество оканчивающихся в правой границе, равной или меньше текущей, отрезков, и далее при фиксации второго валидного отрезка $[a, b]$, прибавить к ответу значение на префиксе $a - 1$ (количество отрезков, имеющих правую границу не больше $a - 1$). $\mathcal{O}(n^2)$ времени.

При решении шестой группы модифицируем решение третьей, чтобы избегать переполнения суммы на отрезке, т.к. она может не влезать в 64-битные типы данных. Поймём, что для фиксированной левой границы валидны лишь первые несколько правых границ, больше левой, так как впервые встретив повтор единичного бита, при добавлении новых чисел к отрезку повтор не исчезнет, а сумма не равна ксору именно в том случае, если есть хотя бы один бит, встречающийся дважды. Потому можно просто фиксировать левую границу и перебирать правую до тех пор, пока сумма равна ксору, и при первом отличии тут же прерывать перебор правой границы.

Решение пятой группы основано на том, что если a_i больше нуля, то с учётом того, что каждый бит на отрезке может быть равен 1 не более одного раза, длина отрезка не превосходит 60. То есть при грамотной реализации, здесь работает решение из 6-й группы, за $\mathcal{O}(n \cdot 60)$.

Для решения на полный балл, нужно модифицировать решение шестой группы. Во первых, для подсчёта валидных отрезков нужно использовать метод двух указателей, ведь при смещении левой границы вправо, максимальная валидная правая граница не могла уменьшиться. Двумя указателями считаем количество отрезков, которые имеют правую границу не больше заданной за $\mathcal{O}(n)$, остаётся оптимизировать перебор второго отрезка. Вместо перебора второго отрезка будем перебирать его левую границу, а количество подходящих к ней правых границ считаем теми же самыми двумя указателями. К ответу прибавим количество правых границ, умноженных на кол-во отрезков, оканчивающихся левее нашей левой границы. Итого $\mathcal{O}(n)$ времени.

Задача E. Два рациональных синуса

По теореме косинусов $\cos\alpha = (a^2 + b^2 - c^2)/2ab$, то есть косинус любого угла треугольника с целыми сторонами обязан быть рациональным. Тем самым проверяем, является ли $\sqrt{1-a^2}$ и $\sqrt{1-b^2}$ рациональным. Если не является, то целочисленных треугольников не существует и ответ -1. Если является, то обозначим их числители за $r_1 = \sqrt{q_1^2 - p_1^2}$ и $r_2 = \sqrt{q_2^2 - p_2^2}$ (знаменатели будут такими же, как у синусов, то есть q_1 и q_2 соответственно).

Синус оставшегося угла выражается через синус и косинус двух заданных, то есть тоже является рациональным, и по теореме синусов отношение сторон рационально, то есть стороны можно домножить на некоторое целое число так, чтобы получился целочисленный треугольник.

Далее для полного решения остаётся аккуратно рассмотреть варианты нетупоугольного или тупоугольного треугольников (случаи, когда третий синус равен $(p_2r_1 + p_1r_2)/q_1q_2$ и $|(p_2r_1 - p_1r_2)/q_1q_2|$ соответственно, если второе значение не обращается в 0), а также не забыть разделить возможный ответ на НОД получившихся сторон после домножения на соответствующий НОК — например, при входных данных 3/5 и 3/5 получаются стороны 15, 15, 24, и после деления на 3 получается правильный ответ 18 (это равнобедренный треугольник, составленный из двух «египетских» треугольников (то есть прямоугольных треугольников со сторонами 3, 4 и 5), объединённых по стороне длины 3).

Если забыть разделить ответ на НОД получившихся сторон, то решение пройдёт только первую группу. Если забыть проверить случай с тупоугольным треугольником, то решение пройдёт первые три группы.

Задача F. Есть бор? Есть два бора!

Обозначим бор Пети как A, бор Васи как B.

Третья подзадача решается любым адекватным полным перебором.

Для решения первой подзадачи нужно понять, что доп. ограничение $p = i$ означает, что бор является бамбуком. Тогда задачу можно переформулировать следующим образом: Даны две строки A‘ и B‘ (равные меткам рёбер в порядке ввода), в запросе нужно ответить, какое максимальное LCP префикса длины $u-1$ строки B‘ и некоторого префикса длины j строки A‘, умноженное на a_j. Но $a_j = 1$, потому ответом на задачу будет LCP строки A‘ и префикса длины $u-1$ строки B‘, а это есть $\min(LCP(A‘, B‘), u-1)$. $LCP(A‘, B‘)$ Нужно вычислить один раз до обработки запросов простым перебором номера символа до тех пор, пока он совпадает. Итого $\mathcal{O}(n+m+q)$ времени работы.

Пусть a_i не обязательно равно 1. Тогда введём массив $b_i = \max(a_i, a_{i+1}, \ldots, a_n)$. Его можно вычислить за $\mathcal{O}(n)$ как $b_i = max(a_i, b_{i+1})$, для всех $i < n$. Тогда верно следующее решение задачи: перебирать длину общего префикса (i) строки A‘ и префикса длины $u-1$ строки B‘ до тех пор, пока символы совпадают, и делать $ans = max(ans, i{*}b_{i+1})$. Не будем делать это на каждом запросе заново, а сохраним в массиве $c_i = max(i * b_{i+1}, c_{i-1})$. Ответом на запрос будет являться $c_{\min(u-1, LCP(A‘,B‘))}$. Итоговое время работы решения $\mathcal{O}(n+m+q)$.

Для решения второй подзадачи вычислим массивы b и c. Теперь нам необходимо найти LCP строки из запроса (конкатенация некоторых префиксов B‘) и строки A‘. Будем по очереди добавлять префиксы из запроса к общему префиксу до тех пор, пока он является таковым. Для проверки на совпадение двух строк можно использовать хеши. Если удалось добавить в конец строк все строки из запроса, и хеш их конкатенации совпадает с хешом соответствующего префикса A‘, то LCP найден. Если же на каком то очередном префиксе u_i его добавление нарушает это равенство, то найдём бинарным поиском добавление какого префикса префикса длины u_i не будет нарушать его. Ответ получаем опять как c_{LCP}. Время работы $\mathcal{O}(n + q \cdot \log m)$.

Обобщим идею из второй подзадачи на бор произвольного вида: массив b_i будем вычислять как максимум значений a_j в поддереве вершины i, а массив $c_i = \max(b_i \cdot len_i, c_{p_i})$, где len_i — длина $s(i)$, а p_i предок вершины i.

Для решения четвёртой подзадачи предпосчитаем значения $s(v)$ для всех вершин v бора B. Теперь можно делать спуск по бору A до тех пор, пока текущая вершина будет префиксом строки из запроса (она легко вычисляется как конкатенация предпосчитанных строк), ответом будет значение c_i в последней достигнутой вершине. Отметим, что в этом решении можно обойтись без массива c_i. Время работы $\mathcal{O}((n+m) \cdot q)$.

В пятой подзадаче можно было объединить два бора в один, например, взять за основу бор

A и добавлять недостающие рёбра из бора B так, чтобы все $s(v)$ из B также были как $s(v)$ в A. Для каждой вершины бора B запомним соответствующую ей вершину из бора A. Тогда ответ на запрос — максимальная глубина предка запомненной вершины, который был исконной вершиной A. Вычислить это можно заранее, в процессе добавления новых рёбер в бор A. Итоговое время работы $\mathcal{O}(n+m+q)$.

В шестой подзадаче аналогично пятой объединим боры, но теперь посчитаем дополнительно массив b, который будем использовать при предпосчёте ответа.

Для полного решения обобщим идею из второй подзадачи. Будем по очереди добавлять строки из запроса, пока будет существовать путь с такими же метками на нём в боре A. Для этого вычислим хеши всех $s(v)$ в обоих борах. Создадим словарь (unordered_map в C++), куда для каждого хеша из бора A добавим значение его c_i. Таким образом, проверка, что при добавлении какой то строки в конец строки запроса существует путь в боре A, сводится к тому, чтобы проверить, есть ли соответствующий хеш в словаре. Остаётся только понять, что делать с бинарным поиском. Для таких случаев на деревьях существуют бинарные подъёмы. Запомним для каждой вершины бора B предка на расстоянии 2^i от него, для всех $0 \leqslant i \leqslant \log_2 n$. Теперь попытаемся подняться на максимальный двоичный подъём от вершины бора B, и поднимемся в том случае, если такая строка всё ещё не будет в словаре. Рассмотрим следующий по убыванию двоичный подъём и сделаем тоже самое и так до нуля. Время работы полного решения $\mathcal{O}(n+(q+m)\cdot\log m)$.

Отметим, что у задачи есть альтернативное решение, не использующее хеши, но работающее в логарифм раз дольше. Основная идея: сделаем аналогию суффиксного массива на двух борах. Чтобы иметь возможность сравнивать двоичные подъёмы между собой, сохраним массивы классов эквивалентности на каждом этапе построения «суффиксного массива». Для каждой вершины бора A для всех степеней двойки запомним в отсортированном списке все классы эквивалентности, достижимые из этой вершины. Тогда будем добавлять не просто строки из запроса по очереди, а добавлять именно их префиксы длины степени 2, и проверять в списке наличие такого перехода.

Задача G. Ёжик играет с графом

Рассмотрим решение для третьей подгруппы.

Назовем все вершины, встречающиеся среди $q_1, q_2, ..., q_k$, *интереснъми.*

Пусть deg_u – входящая степень вершины u.

Пусть мы сделали первый выбор во всех предложениях и увеличили ответ на соответствующую величину. Далее будем говорить, что предложение *применено*, если мы поменяли в нем выбор на второй.

Для каждой интересной вершины u выпишем все входящие в нее вершины v и найдем для каждой из входящих вершин величину up, равную тому, сколько нужно добавить к b_u, чтобы вершина u стала наибольшим(нестрого) соседом v по значениям b. Затем отсортируем набор значений up, домножим все значения на -1 и превратим его в массив префиксных сумм. Получим набор $up(u,i)$. Значение $up(u,i)$, таким образом, означает, на сколько нужно увеличить b_u, чтобы u стала наибольшим соседом у i входящих в нее вершин.

Давайте посчитаем сначала начальный ответ без применения предложений.

Теперь, пусть в оптимальном решении интересная вершина u дает неотрицательный вклад по i входящим в нее рёбрам.

Пусть для предложения (p,q,s,r) $X = s\cdot c_p$, $Y = r$. Тогда можно сказать, что предложения, для которых если $i\cdot Y - X < 0$, не применялись.

Что, если мы сначала зафиксируем i у некоторой интересной вершины q, а затем применим те предложения, в которых $i\cdot Y - X \geqslant 0$, сделаем так для всех i и возьмем максимум? Обозначим за $score(u,i)$ величину, которую мы получим таким алгоритмом для i дающих вклад соседей, пусть $score(u,0) = 0$.

Может так оказаться, что u на самом деле не является максимальным соседом для i-й входящей вершины, однако это может произойти только в том случае, если $\sum Y < up(u,i)$ по примененным предложениям.

Однако, $\sum(i\cdot Y - X)$ - $\sum\max(0,(i-1)\cdot Y - X) \leqslant \sum Y < up(u,i)$, здесь суммирование происходит по примененным для i предложениям; поэтому ответ для $i-1$ будет лучше ответа для i, и мы не

испортим глобальный ответ тем, что обновим его величиной $score(u, i)$. Таким образом, нужно взять $\max_{i=0}^{deg_u} score(u, i)$ и сложить эти значения для всех интересных вершин u.

Получаем следующий алгоритм: идем по предложениям, находим $i = \lceil \frac{X}{Y} \rceil$ и прибавляем на суффиксе $[i, deg_u]$ величину $j \cdot Y - X$ для $i \leqslant j \leqslant deg_u$, здесь deg_u – входящая степень вершины u. Прибавления можно делать в цикле.

Данное решение имеет сложность $O(m \cdot (\log m + G) + n + k)$.

Перейдем к полному решению задачи.

Пусть теперь есть запросы.

Важной частью решения является то, что все прибавления d положительны. , Это значит, что X в предложениях может только расти.

Поэтому индекс суффикса $score(q)$, на котором предложение будет положительно, будет только увеличиваться.

Апдейт для q будет выглядеть следующим образом: отменить прибавление $i \cdot Y - X$ на некотором отрезке(так как оно стало меньше нуля), после чего вычесть константу на суффиксе. При этом, если рассмотреть величины, которые мы вычитаем для данного q(это $i \cdot Y - X$ и константа на суффиксе), то с увеличением i эти величины только растут.

Таким образом, у нас есть события вида «увеличение номера суффикса, на котором предложение положительно». Так как каждая q в предложениях встречается не более G раз, суммарно будет не более $n \cdot G$ увеличений суффикса.

Мы можем поддерживать для каждого p некоторую структуру данных(например, сет), с помощью которой отслеживать изменения на суффиксах.

Также, есть предложения, в которых не меняется суффикс, но может поменяться максимум – если он был на суффиксе, то перейти в префикс после вычитания константы. Такое тоже будем отслеживать структурой данных. Таких событий может быть не более $O(n)$.

По предложениям, в которых не произошли события, храним некую дополнительную информацию, с помощью которой можно пересчитывать ответ без прохода по таким предложениям.

Чтобы быстро делать изменения в точке и вычитания на суффиксе в предложениях, будем по каждому $score(q)$ хранить дерево отрезков с массовыми изменениями.

Стоит отметить, что реализация получается непростая за счет того, что нужно хранить и учитывать немало вспомогательных величин, чтобы следить за предложениями, в которых не произошло событий в данный момент.

Сложность решения – $O(m \cdot \log m \cdot G + n + k + q)$.