Задача A. Буквы на заказ

Если внимательно изучить строчные буквы алфавита, то можно увидеть, что есть только

- одна буква с двумя отверстиями:

  `<<B>>`

- шесть букв с одним отверстием:

  `<<A>>, <<D>>, <<O>>, <<P>>, <<Q>>` и `<<R>>`

Все остальные буквы не имеют отверстий.

В этой задаче важно смотреть именно на список всех букв из условия задачи, так как количество отверстий может отличаться в зависимости от шрифта.

Задача B. Бариста

Если выполнено условие $a : b < 3 : 1$, то необходимо вывести слово «macchiato». Для того, чтобы избавиться от вещественного деления, преобразуем условие к виду $a < 3 \cdot b$.

Если выполнено условие $a : b > 5 : 1$, то необходимо вывести слово «latte». Аналогично, чтобы избавиться от вещественного деления, преобразуем данное условие к виду $a > 5 \cdot b$.

Если не выполнилось ни первое, ни второе условие, то следует вывести слово «cappuccino».

Задача C. Посудомойка

Заведём массив q, в котором $q_i = 1$, если среди запросов есть запрос вида $(1, i)$, иначе $q_i = 0$. В каждой из стопок найдём самую нижнюю тарелку, у которой $q_i = 1$. Помоем i-ю тарелку, а также все тарелки, которые лежали над ней.

С одной стороны, в стопках останутся только те тарелки, которые никогда не будут выставляться на стол. С другой стороны, все тарелки, которые хотя бы раз будут выставляться на стол, окажутся в шкафу.

Далее будем действовать по следующему алгоритму:

- На запросы первого типа будем брать уже помытую тарелку из шкафа.
- На запросы второго типа будем складывать тарелку в первую стопку и смотреть, будет ли она когда-либо использоваться в будущем. Если да, то сразу же достанем её и помоем, если нет, то оставим лежать в стопке.

  Для того, чтобы эффективно определять, будет ли тарелка использоваться в дальнейшем, заведем дополнительный массив e: $e_i = j$, если $q_i = 1$ и самый поздний запрос $(1, i)$ имеет номер j во входных данных. В ином случае $e_i = -1$.

  В таком случае проверка сводится к сравнению величины e_i и номера текущего запроса.

Таким образом, к моменту каждого запроса первого типа, необходимая тарелка уже будет в шкафу.

Такой алгоритм позволяет достичь минимального количества операций, потому что он не делает «лишних» действий:

- Если $q_i = 0$, и в стопке под тарелкой i нет тарелок $j : q_j = 1$, то тарелку i ни разу не помыли.
- Если $q_i = 0$, и в стопке под тарелкой i есть тарелка $j : q_j = 1$, то тарелку i помыли ровно один раз. При этом её нельзя ни разу не помыть, потому что она «загораживает» тарелку j.
- Если $q_i = 1$, и всего было t запросов $(1, i)$, то тарелку i помыли ровно t раз. При этом невозможно ответить на все просьбы мамы, помыв тарелку i меньшее количество раз.

Задача D. Королевства и союзы

Обозначим текущую религию i-го королевства как a_i.

Для начала рассмотрим решение задачи при $\mathbf{k = 1}$.

В таком случае потенциальным союзом является любой отрезок $[L \dots R]$ такой, что $a_L = a_{L+1} = \dots = a_R$.

Назовем отрезок $[L \dots R]$ полным, если выполняются все перечисленные ниже условия:

- отрезок является потенциальным союзом ($a_L = \dots = a_R$);
- $L = 1$ или $a_{L-1} \neq a_L$;
- $R = n$ или $a_{R+1} \neq a_R$.

В таком случае массив a разбивается на множество непересекающихся полных отрезков $[L_1 = 1 \dots R_1]$, $[L_2 = R_1 + 1 \dots R_2]$, $\dots$, $[L_k = R_{k-1} + 1 \dots R_k = n]$.

Рассмотрим, как разбиение на полные отрезки определяет количество потенциальных союзов:

- Любой потенциальный союз является подотрезком одного из полного отрезков в разбиении массива;
- Количество потенциальных союзов внутри полного отрезка $[L \dots R]$ равно $C(R - L + 1, 2)$, где $C(M, 2) = \frac{M \cdot (M-1)}{2}$;
- Суммарное количество союзов равно сумме по всем полным отрезкам в разбиении.

Легко заметить, что изменение религии в королевстве i может изменить разбиение массива одним из следующих способов:

- Королевство i было строго внутри полного отрезка — после изменения полный отрезок превратился в 3 (слева от i, само i, справа от i).
- Королевство i было на краю полного отрезка — в таком случае оно могло или образовать новый полный отрезок единичной длины, или объединиться с соседним полным отрезком (если у них совпали религии после изменения).

Получается, что если получится эффективно поддерживать длины отрезков в разбиении, то можно будет пересчитывать итоговое количество союзов за $O(1)$ — вычитая из ответа количества союзов в затронутых изменением «старых» отрезках и прибавляя к ответу количества союзов в «новых» отрезках.

Для того, чтобы эффективно поддерживать разбиение, нам понадобится структура данных, реализующая следующие операции:

- Найти отрезок, в котором в данный момент находится королевство i;
- Удалить отрезок;
- Добавить отрезок.

Легко понять, что в качестве такой структуры можно использовать словарь на основе двоичного дерева поиска, где ключом является конец отрезка, а значением — его начало.

В таком случае операции «добавить» и «удалить» являются базовыми операциями словаря «добавить (ключ, значение)» и «удалить ключ».

Поиск же отрезка, содержащего данное королевство, можно осуществлять с помощью операции двоичного дерева поиска «найти ближайший больший ключ».

Все описанные выше операции выполняются за $O(\log n)$, поэтому суммарная сложность будет $O(q \cdot \log n)$.

Но всё это время мы решали задачу для $k = 1$. Как изменится решение при $\mathbf{k > 1}$?

На самом деле никак. Легко заметить, что все королевства разбиваются на независимые группы по величине $i \ mod \ k$, где mod — операция взятия остатка от деления.

Внутри каждой группы королевств действует все вышесказанное про решение с $k = 1$, а итоговый ответ является суммой по всем различным группам. Соответственно, теперь вместо одного словаря необходимо просто хранить k различных словарей и модифицировать каждый раз только соответствующий группе измененного королевства.

Задача E. Стикеры

Заведем массив cnt размера N: в элементе cnt_i будет записано, сколько человек указало i-го участника как пригласившего.

Переберем всех участников от 1 до N. Проверим, что i-й участник — «хороший» друг.

- Введем обозначение $f = F_i$. Если $f \neq 0$, то участник i может сделать своего друга f на шаг ближе к получению стикера;
- Если $P_i > 0$, то участник i прошел проверку на решение задач.
- Если $T_i > T_f$, то участник зарегистрировался позже своего друга.
- Если все проверки пройдены, то увеличим cnt_f на 1, т.е. засчитаем данного участника как друга, удовлетворяющего всем условиям.

После обработки всех участников еще раз переберем всех участников для восстановления итогового ответа. Участник i заслуживает стикер, если $cnt_i \geqslant 2$.

Осталось не забыть вывести всех участников в порядке возрастания их номеров — этого легко добиться, если сохранить всех участников в список в порядке их перебора от 1 до N.

Задача F. Прыгай вперед!

Можно показать, что путь от клетки 1 до клетки n всегда существует и единственный. Действительно, в какой бы клетке мы не находились, всегда есть ровно один способ прыгнуть в клетку с большим номером. Поэтому, прыгая вперед, в какой-то момент мы достигнем клетки с номером n.

Таким образом, нет необходимости искать минимальный путь, достаточно уметь находить стоимость единственного имеющегося пути из клетки 1 в клетку n.

Для решения используем принцип корневой декомпозиции. Разобьем клетки на группы размера k. В первую группу войдут клетки с номерами $1, 2, \ldots, k$, во вторую группу войдут клетки с номерами $k+1, k+2, \ldots, 2 \cdot k$, и так далее. В последней группе может оказаться меньше чем k клеток, поэтому последнюю группу нужно рассматривать аккуратно или дополнить ее фиктивными элементами так, чтобы все группы имели размер в точности k клеток.

После любого количества изменений у каждой клетки однозначно определена величина прыжка вперед. Эту величину нужно поддерживать для каждой клетки, это можно делать за $O(1)$.

Дополнительно для каждой клетки i будем поддерживать дополнительную информацию. Пусть мы начали прыгать из клетки i вперед до тех пор, пока не попали в другую группу клеток. Это может быть следующая группа клеток или группа находящаяся еще дальше. Пусть клетка, в которую мы в итоге попали, имеет номер $to[i]$, а $co[i]$ — это сумма стоимостей всех клеток, которые мы посетили, прыгая от клетки i до клетки $to[i]$, за исключением клетки i. Тут нужно аккуратно разобраться с тем, как определяются величины $to[i]$ и $co[i]$ для клеток последней группы, возможен вариант при котором $to[i]$ — клетка с номером n или вариант, при котором добавляется еще одна фиктивная группа, куда мы попадаем из последней группы.

Поддерживая величины $to[i]$ и $co[i]$ мы можем после любого количества изменений вычислить стоимость пути из 1 в n за $O(n/k)$. Для этого стартуем из клетки $i = 1$ и делаем переходы из i в $to[i]$ до тех пор, пока не пройдем весь путь, суммируя величины $co[i]$. Важно не забыть прибавить к полученному результату стоимость посещения клетки 1. Так как каждый переход из i в $to[i]$ меняет текущую группу, то количество переходов не будет превышать количество групп, то есть все переходы будут сделаны за $O(n/k)$.

После каждого применения карточки необходимо пересчитывать все величины, которые могли измениться. Пусть карточка изменила размер прыжка из клетки i. Тогда величины to и co не изменятся для клеток с номерами больше i. Для клеток из других групп, отличных от группы клетки i, также останутся неизменными величины to и co. Следовательно, величины to и co нужно пересчитать только для тех клеток, которые лежат в одной группе с клеткой i и имеют номера не больше i. Если выполнять пересчет справа налево, то есть начиная от клетки i, то на пересчет величин to и co потребуется $O(k)$ действий.

Если выбрать размер группы равным $k = \sqrt{n}$, то после применения карточки пересчет величин to и co, а также вычисление стоимости пути от 1 до n будет выполняться за $O(\sqrt{n})$, а вся задача будет решена за $O(q\sqrt{n})$ действий.

Задача G. Полив... «Ой!»

Сначала разберём крайние случаи:

- $A = B$. Можно не делать никаких операций преобразования.
- B состоит из всех единиц, $A \neq B$. Тогда в A есть хотя бы один ноль. Заметим, что операция (i, l, r) делает i-й бит равный нулю. Поэтому после каждого преобразования над строкой A в ней остаётся хотя бы один нулевой бит. Значит невозможно получить B из A.
- A состоит из всех нулей, $A \neq B$. Тогда над A нельзя выполнить ни одной операции, поэтому невозможно получить B из A.

Далее считаем, что $A \neq B$, в A есть хотя бы один единичный бит, в B есть хотя бы один нулевой бит.

Заметим, что если i-й бит в A равен единице, то можно сделать операцию (i, i, i) и занулить данный бит. Поэтому, если бы можно было преобразовать A в строку из всех единиц, то мы бы её преобразовали к такой строке, а затем выполнили операции (i, i, i) для всех $i : B_i = 0$. Получили бы $A = B$. Но, к сожалению, получить строку из всех единиц невозможно.

Тогда придумаем алгоритм получения строки из «почти всех» единиц, а именно: найдём позицию $z : B_z = 0$, и придумаем, как получить строку, у которой на всех позициях, кроме z-й, стоят единицы, а на z-й позиции стоит ноль. А затем «занулим» некоторые биты A так, чтобы получить $A = B$. В итоге получим такой алгоритм:

1. Зафиксируем произвольный индекс z, такой что $B_z = 0$.
2. Хотим, чтобы A_z было равно 1. Если это не так ($A_z = 0$), то найдём ближайшую слева от z позицию $x : A_x = 1$ и сделаем операцию (x, x, z). Если такой позиции x слева от z не существует, то найдём позицию $x : A_x = 1$ справа от z и сделаем операцию (x, z, x).
3. Занулим все биты в A, кроме z-го, используя операции вида (i, i, i). Получим строку из всех нулей и одной единицы на позиции z.
4. Сделаем запрос $(z, 1, n)$, тем самым получим строку из всех единиц и одного нуля на позиции z.
5. На этом этапе $A_z = B_z = 0$, а все остальные биты в строке A единичные. Тогда для всех $i : B_i = 0$ сделаем операцию (i, i, i) и получим $A = B$.

Оценим количество операций в таком алгоритме:

- Если $A_z = 1$, то потребуется не более $n - 1$ операций на то, чтобы было верно $A_i = 1 \Leftrightarrow i = z$.
- Если $A_z = 0$, то потребуется одна операция, чтобы сделать $A_z = 1$ и не более $n-2$ операций на то, чтобы было верно $A_i = 1 \Leftrightarrow i = z$ (всего n позиций, при этом не надо применять операции к позиции z и к позиции x, так как $B_x = 0$ после первой операции). Потратили не более $n - 1$ операций.

- Одна операция $(z, 1, n)$.
- Не более $n-1$ операций на то, чтобы «занулить» некоторые биты в A.
- Всего потратили не более $(n-1)+1+(n-1) = 2 \cdot n - 1$ операций.

В задаче ограничения на количество операций были $4 \cdot n+1$, поэтому допускались и другие менее оптимальные решения.

Задача H. Дорога в школу.

Решение за 3 запроса.

Можно сделать три запроса типа «?» для участков дороги, расположенных примерно на одинаковом расстоянии друг от друга. Например, для участков дороги с номерами 1, $[\frac{n}{3}]+1$ и $[\frac{2 \cdot n}{3}]+1$, где $[x]$ обозначает целую часть числа x.

Так как кратчайший путь от дома Пахома до школы содержит не более половины участков дороги, то он не может содержать в себе все три участка из запросов. Следовательно, ответ хотя бы на один из запросов будет в точности равен искомой длине кратчайшего пути. Если участок дороги из запроса лежит на кратчайшем пути, то ответ на этот запрос будет не меньше искомой длины. Из этого следует, что ответ на задачу можно получить как минимум из ответов на наши три запроса.

Решение за 2 запроса.

Аналогично предыдущему решению можно поступить, используя только два запроса типа «?». Например, можно сделать запросы для участков дороги с номерами 1 и $[\frac{n}{2}]+1$.

Решение за 1 запрос.

Пусть x — это длина кратчайшего пути. Тогда ответом на любой запрос для участка дороги, который не лежит на кратчайшем пути, будет число x. А ответом на любой запрос для участка дороги, который лежит на кратчайшего пути, будет число $n-x$. При этом выполнено: $x \leqslant \frac{n}{2}$, $n - x \geqslant \frac{n}{2}$.

Теперь понятно, что достаточно одного запроса типа «?» для решения задачи. Сделаем этот запрос, например, для участка дороги с номером 1 (подойдет и любой другой участок дороги).

Пусть ответ на этот запрос равен k. Если $k \leqslant \frac{n}{2}$, то это и есть длина кратчайшего пути. Если же $k > \frac{n}{2}$, то длина кратчайшего пути равна $n-k$. Можно даже не разбирать эти два случая, а просто выводить в качестве ответа величину $\min(k, n-k)$.

Задача I. Башни из спичек

В данной задаче надо рассмотреть 2 случая:

1. Башня состоит из квадратов
2. Башня состоит из прямоугольников

Для решения обоих случаев нам понадобится словарь (или массив, т.к. $A_i \leqslant 2 \cdot 10^5$), в котором ключом будет длина спички, а значением — количество спичек такой длины. Назовем его cnt, где cnt_i — количество спичек длины i.

Рассмотрим первый случай. Переберем каждую длину спички i и посмотрим, какая получится башня из спичек только этой длины. Количество этажей F_i, которое получится составить из спичек длины i равняется $[\frac{cnt_i - 1}{3}]$ (где $[x]$ — целая часть числа x). Итоговая высота башни из квадратов будет равна $i \cdot F_i$.

Доказательство формулы для квадратов: пусть в башне F этажей. В таком случае у башни есть фундамент из одной спички, а каждый новый этаж добавляет две стены и один потолок, суммарно $1 + 3 \cdot F$ спичек.

Рассмотрим второй случай — в башне есть спички-стены длины h и спички-потолки длины w. Максимальное количество этажей в таком случае равно $\min([\frac{cnt_h}{2}], cnt_w - 1)$ (доказательство формулы аналогично формуле с квадратами).

Для поиска ответа переберем высоту стен h в ответе. Какую ширину следует выбрать при фиксированной высоте стен? На ответ данный выбор может повлиять только в том случае, если нам не

хватит спичек-потолков. Поэтому выберем в качестве ширины такую длину спички, которая чаще всего встречается среди представленных.

Так как «наиболее часто встречающаяся длина» не зависит от выбора высоты башни, то её можно предпросчитать заранее.

Осталось рассмотреть лишь один дополнительный случай — длина спички, которую мы использовали для потолков, выбрана в качестве высоты стен башни. В таком случае в качестве длины спичек-потолков можно использовать вторую по количеству встречаемости во входных данных длину. Или можно перебрать все имеющиеся длины и сравнить получающиеся ответы для каждой.

Итоговая сложность решения $O(N + A)$, где $A = 2 \cdot 10^5$ — максимальная длина одной спички.