Задача A. Бег в две стороны

Пусть расстояние до очередного пункта будет x, тогда чтобы добежать до него и вернуться, нужно $2 \cdot x$ секунд. Так нужно сделать для всех пунктов, кроме одного. Очевидно, чтобы минимизировать количество секунд, нужно не бежать дважды до самого дальнего пункта.

Ответом будет $2 \cdot sum(l) + 2 \cdot sum(r) - max(max(l), max(r))$.

Задача B. Бег в одну сторону

Заметим, что ответ на задачу не превосходит v_2^2 — за первые v_2 секунд человек пробежит $v_2 \cdot v_1$ метров, после чего робот наберет максимальную скорость $v_2 > v_1$ и каждую секунду будет бежать быстрее человека.

Тогда для решения первой группы тестов ($v_2 \leqslant 10^3$) достаточно промоделировать позиции робота и человека, для этого потребуется на более 10^6 итераций.

Для решения второй группы тестов ($v_2 \leqslant 10^6$) надо промоделировать первые v_2 секунд. После этого можно найти расстояние d — насколько робот отстает от человека (если еще не обогнал его в процессе моделирования). Тогда робот должен пробежать хотя бы на $d+1$ метр больше, чем человек, для этого потребуется $(d+1)/(v_2 - v_1)$ секунд, округленное вверх.

Для решения полной задачи ($v_2 \leqslant 10^9$) можно воспользоваться бинарным поиском в диапазоне от 1 до v_2, чтобы найти первый момент времени, когда робот обогнал человека. Для этого достаточно посчитать, что в момент времени t робот пробежит $t \cdot (t+1)/2$ метров, а человек — $t \cdot v_1$ метров (и он был на 1 метр впереди изначально). Если позиция робота правее человека, то надо двигать правую границу поиска, иначе — левую. Если же в процессе поиска ни разу не случилось, что робот обогнал человека, то, значит, ему сначала надо набрать максимальную скорость, после чего можно воспользоваться формулой из предыдущей подзадачи.

Задача C. Плакат

Для решения первой группы тестов ($n \leqslant 20$) достаточно перебрать, где может располагаться одно название или два названия — каждое из них состоит из 7 букв, поэтому 3 и более надписи NEIMARK не поместятся на плакате.

Во второй группе можно заметить, что из слова NEIMARK всегда совпадают максимум 3 буквы — гласные ‘E’, ‘I’, ‘A’. Тогда единственный случай, когда стоит писать это название — это наличие всех гласных на своем месте. В этом случае стажер заработает 1 очко. Поэтому здесь работает жадный алгоритм, двигаясь по исходной строке слева-направо, нужно заменить буквы на NEIMARK, начиная с текущей позиции, как только 3 из них совпадут. При этом важно учесть, что текущую позицию надо сдвинуть сразу на 7, чтобы не изменить буквы уже написанного слова NEIMARK.

Для решения полной задачи можно воспользоваться динамическим программированием (‘изи ДП’). Пусть $dp[p]$ — максимальное количество очков, которые можно заработать, изменяя буквы плаката, начиная с позиции p. Будем перебирать p, начиная со значения n. Во-первых, можно не менять текущую букву и перейти сразу к следующей, то есть значение можно инициализировать так:

$$dp[p] = dp[p+1]$$

Если же мы хотим написать надпись NEIMARK с текущей позиции, то надо посчитать количество несовпадающих букв d и тогда ответ можно обновить следующим образом:

$$dp[p] = max(dp[p], dp[p+7] + 5 - d)$$

Посчитанное таким образом значение $dp[0]$ и будет ответом на задачу.

Задача D. Линия метро

Получившееся строение города — дерево, где корнем является центральный квартал.

Для подгруппы 1 достаточно перебрать каждую пару вершин — поднимаемся от пары вершин к корню дерева и считаем суммы вершин, запоминаем, сколько вершин прошли, если в какую-то вершину зашли дважды — значит, вершины не подходят, иначе, если посещенных вершин не больше k, то обновляем максимальную сумму.

Для подгруппы 2 также перебираем пары вершин, только нужно научиться быстро находить сумму от вершины до корня, расстояние до корня и проверять, не посетим ли какую-то вершину дважды. Сумму можно находить как $sum_v = sum_u + a_v$, расстояние $d_v = d_u + 1$ для пары вершин, где u — предок вершины v. Чтобы не посещать какую-то вершину дважды — нужно брать две вершины из разных поддеревьев, исходящих от корня(тогда корнем каждого такого поддерева будет вершина, связанная с вершиной 1), для каждой вершины можно хранить такую вершину, заведем массив pv_i — если $p_v = 1$, то $pv_v = v$, если $p_v = u$, то $pv_v = pv_u$, так для каждой вершины мы запомним, в каком поддереве она находится. Теперь для пары вершин v, u, если $pv_v = pv_u$ или $d_v + d_u + 1 > k$, то такую пару проверять нет смысла.

Для подгруппы 3 нужно найти две самые большие суммы в разных поддеревьях, так как ограничения позволяют нам выбрать любую пару вершин. Можно запоминать суммы и корни поддеревьев как в подгруппе 2. Только теперь нам не нужно проверять пары вершин, нужно циклом пройтись по всем вершинам и найти вершину v с максимальной суммой sum_v, затем таким же образом вершину u с максимальной суммой sum_u, такой что $pv_v \neq pv_u$.

Для полного решения также посчитаем все sum_v, pv_v, d_v для каждой вершины v. Давайте заведем массив max_d, где max_d_k — максимальная сумма на расстоянии не более k от корня дерева, изначально для всех k значение $max\_d_k = 0$. Теперь будем проходить каждое поддерево последовательно от расстояния 1 до k, тогда в момент посещения вершины v ответ может обновиться как $ans = max(ans, sum_v + max\_d_{k-d_v})$, т.е. смотрим на сумму до вершины v, а также на максимальную сумму на расстоянии не более $k - d_v$ от корня из уже пройденных поддеревьев. Затем еще раз проходимся по вершинам из поддерева и обновляем $max\_d_{d_v} = max(max\_d_{d_v}, max\_d_{d_v-1}, sum_v)$, если на расстоянии d_v мы нашли большую сумму, чем из предыдущих поддеревьев.

Задача E. Взаимно-упрощенные

Для решения первой группы тестов ($n \leqslant 100$) достаточно перебрать все пары чисел, для каждой найти наибольший общий делитель (например, воспользовавшись стандартным алгоритмом gcd в C++) и проверить, является ли этот делитель простым. Даже неэффективная проверка на простоту позволит пройти тесты этой группы, сложность решения — $O(n^2 \cdot \sqrt{n})$ или $O(n^3)$ в зависимости от реализации проверки на простоту.

Во второй группе ($n \leqslant 3000$) уже потребуется реализовать эффективную проверку чисел на простоту, например, при помощи решета Эратосфена. Этот алгоритм позволит сразу найти все простые числа, не превосходящие n, после чего проверку можно выполнить за $O(1)$, а общая сложность решения составит $O(n^2 \cdot ln(n))$.

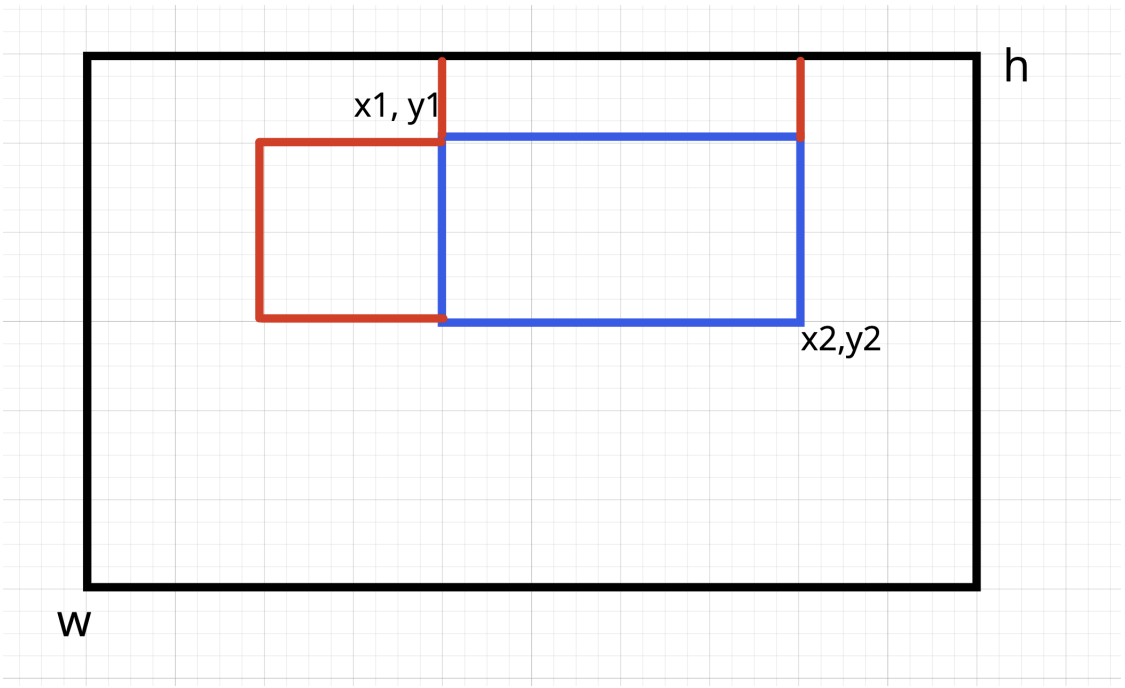
Чтобы еще ускорить решение и пройти следующую группу тестов ($n \leqslant 10000$) необходимо ускорить вычисление наибольшего общего делителя, чтобы избавиться от логарифма в асимптотике. Это можно сделать, например, при помощи динамического программирования — для каждой пары чисел $1 \leqslant x \leqslant y \leqslant n$ сохранить значение (x, y), находя его через ровно один шаг алгоритма Евклида — $(x, y) = (y, x\%y)$ (где % — взятие остатка от деления).

Для прохождения следующей группы тестов ($n \leqslant 5 \cdot 10^5$) уже потребуется новый подход. Давайте переберем, чему будет равен наибольший общий делитель интересующих нас пар чисел. Это должно быть какое-то простое число p, а сами числа, соответственно, должны делиться на p. Таких чисел всего n/p и они имеют вид $k \cdot p$, где k — натуральное число $1 \leqslant k \leqslant n/p$. Тогда у интересующих нас пар чисел соответствующие значения k должны быть взаимно простыми. Тогда количество взаимно-упрощенных пар для конкретного значения p будет равно сумме значений функции Эйлера для чисел от 1 до n/p. Итого, для решения потребуется найти значения функции Эйлера для всех чисел от 1 до n, например, путем разложения на простые множители (асимптотическая сложность — $O(\sqrt{n})$ для каждого числа), затем найти префикс-суммы полученного массива и ответ для каждого числа p будет вычисляться за $O(1)$. Общая сложность — $O(n \cdot \sqrt{n})$.

Наконец, для решения полной версии задачи потребуется воспользоваться предыдущим подходом, но ускорить вычисление функции Эйлера. Для этого можно воспользоваться решетом Эратосфена, но для каждого числа записать не флаг простое/составное, а сохранить один из простых делителей. Тогда разложение на простые множители и, соответственно, нахождение функции Эйлера можно будет выполнить за $O(ln(n))$. Таким образом, общая сложность решения составит $O(n \cdot ln(n))$.

Задача F. Змейка

Если у нас есть препятствие с координатами $(x_1, y_1), (x_2, y_2)$ и змейка длины k, тогда змейка не может начинать в любой точке в прямоугольнике $(x_1, y_1), (x_2, y_2)$, а также еще в прямоугольниках $(x_1 - k + 1, y_1), (x_1 - 1, y_2)$(если змейка горизонтальная) и $(x_1, y_1 - k + 1), (x_2, y_1 - 1)$(если змейка вертикальная) (тк если змейка будет начинаться в них, то часть змейки зайдет на прямоугольник $(x_1, y_1), (x_2, y_2)$).



Подгруппа 1:

У нас всего 1 препятствие, по принципу выше — нужно вычесть все клетки трех прямоугольников, можно решить при помощи формулы $w \cdot (h-k+1) - S(x_1, y_1, x_2, y_2) - S(x_1-k+1, y_1, x_1-1, y_2) + (w-k+1) \cdot h - S(x_1, y_1, x_2, y_2) - S(x_1, y_1-k+1, x_2, y_1-$ где S - площадь прямоугольника. Стоит учесть, что x_1-k+1, y_1-k+1, x_1-1 или y_1-1 может быть меньше 1, это означает, что прямоугольник выйдет за границы исходного $(1, 1), (w, h)$, еще нас не интересуют последние $k-1$ столбцов при горизонтальном расположении змейки, а также если длина змейки $k = 1$, то мы каждый вариант посчитали дважды.

Подгруппа 2:

Решение в лоб — заводим матрицу и закрашиваем все клетки, где есть препятствия, затем для каждой линии находим расстояния между соседними препятствиями, тогда к ответу прибавляем $max(0, x_2 - x_1 - k)$, аналогично для столбцов.

Подгруппа 3:

Если у нас будет 1 строка, то задача сводится к одномерной — найти пересечения отрезков, а затем расстояния между получившимися отрезками, к ответу прибавляем $max(0, x_2 - x_1 - k)$.

Пересечь отрезки можно, например, отсортировав их по левой границе, а затем пройтись методом двух указателей, поддерживая левую и правую границы текущего отрезка.

Аналогично, если у нас будет 1 столбец.

Подгруппа 4:

Пусть k равняется количеству столбцов, тогда чтобы расположить змейку на строке — нужно, чтобы все столбцы были свободны. Значит, если в строке есть хотя бы 1 прямоугольник, то расположить нельзя. По сути задача сводится к предыдущей группе — нужно взять все отрезки для строк, пересечь их, а затем найти расстояния между соседними x_2-x_1-1(это будет количество пустых строк), аналогично если k равняется количеству строк .

Подгруппа 5:

Препятствия - это квадраты размером 1 на 1(а значит можно представить, что это единичные уже пересеченные отрезки, и сведем задачу к подгруппе 3), давайте отсортим их по строкам(вторым приоритетом по столбцам). Затем будем проходиться по ним и смотреть расстояния между столбцами, если номера строк совпадают, то к ответу прибавляем $max(0, x_2 - x_1 - k)$.

Подгруппа 6:

Если длина змейки равна 1, то нужно найти площадь объединения всех препятствий и вычесть ее из площади поля. Можно сделать при помощи ДО.

Подгруппа 7:

Если у нас есть препятствие с координатами $(x_1, y_1), (x_2, y_2)$ и змейка длины k, тогда змейка не может начинать в любой точке в прямоугольнике $(x_1, y_1), (x_2, y_2)$, а также еще в прямоугольнике $(x_1 - k + 1, y_1), (x_1 - 1, y_2)$(если змейка горизонтальная), по сути в одном прямоугольнике $(x_1 - k + 1, y_1), (x_2, y_2)$.

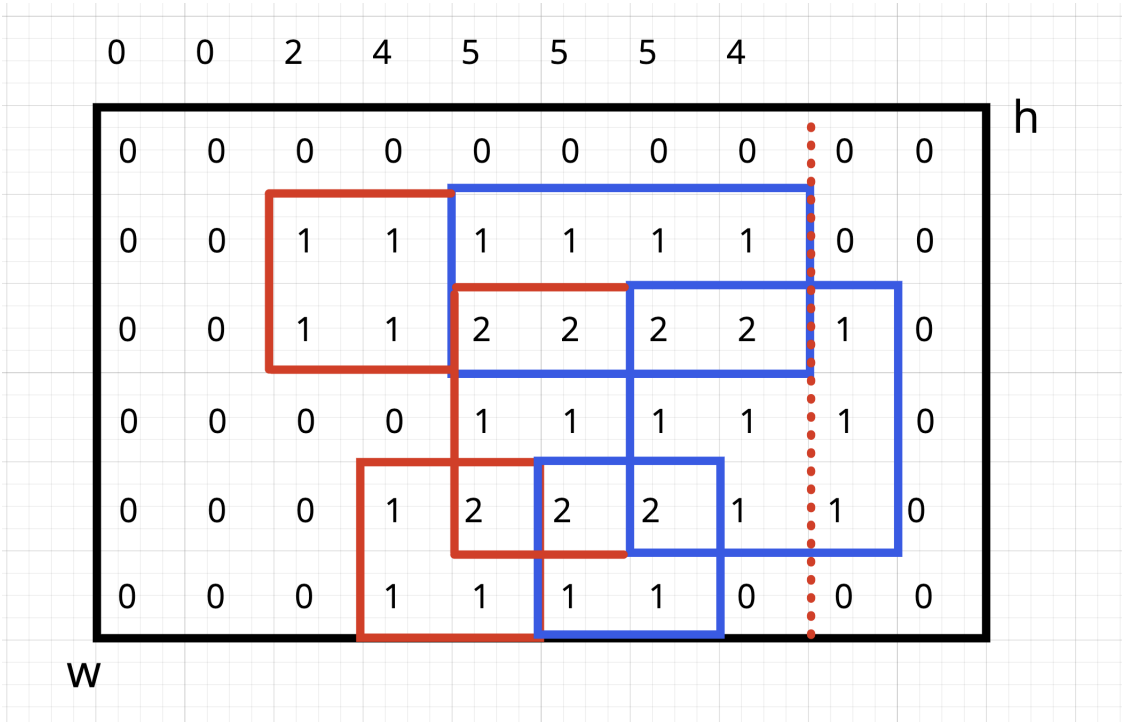
Давайте в таком случае, когда мы находимся на y_1, то ко всему отрезку $x_1 - k + 1..x_2$ прибавим 1, а когда будем на координате $y_2 + 1$, то на этом же отрезке $x_1 - k + 1..x_2$ убавим 1. Тогда в какой-то момент времени y мы возьмем отрезок $1..w$, то количество ненулевых элементов — количество клеток, где нельзя расположить змейку.

Получается, что в каждом препятствии нас интересуют по два отрезка — где начинается область, куда нельзя змейку поставить и где она заканчивается.

Тогда давайте отсортим все получившиеся отрезки по y, затем для всех одинаковых y выполним запросы прибавления и убавления на отрезке, после этих операций — кол-во ненулевых элементов на всех строках будет количеством недопустимых клеток на y, считаем их для каждого y от 1 до h, а затем вычитаем из общего количества клеток $w \cdot h$.

Можно завести Дерево Отрезков и выполнять операции на отрезке $1..10^6$.

Пример для змейки длиной 3 (синяя область — препятствия, красная — область, куда нельзя ставить змейку). Также нас вообще не интересуют столбцы с номером больше $h - k$, тк на них попросту нельзя поставить горизонтальную змейку:



Аналогично для вертикального расположения змейки.

Полное решение:

Полное решение аналогично подгруппе 7, только чтобы выполнять операции в ДО нужно сжать координаты, тк мы не сможем хранить ДО с 10^9 элементов.