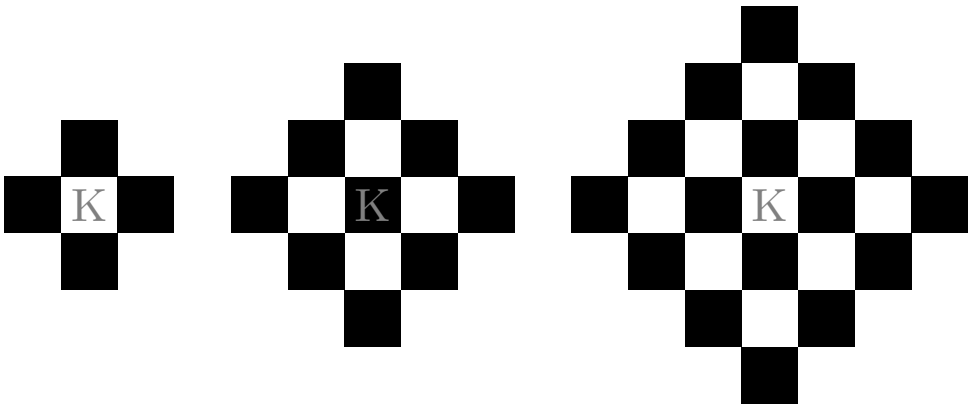
Задача 1. Камень в море

Рассмотрим рисунки для $T = 1$, $T = 2$ и $T = 3$, приведенные ниже. Внимательно посмотрев на них, уже можно увидеть закономерность: количество клеток, занятых волнами спустя T секунд после броска камня, равно $(T + 1)^2$: на каждом рисунке изображены $T + 1$ диагоналей, в каждой из которых находятся $T + 1$ черных клеток.



Однако, можно строго доказать нашу догадку. Заметим, что в зависимости от четность T центральная клетка либо содержит, либо не содержит волны. Докажем по индукции, что ответ равен $(T + 1)^2$ для четных и нечетных T по-отдельности.

Рассмотрим нечетные значения T. База индукции: $T = 1$, ответ равен $(T + 1)^2 = 4$. Выполним переход индукции. Пусть для некоторого T ответ равен $(T + 1)^2$. Докажем теперь, что спустя $T + 2$ секунд после броска камня ответ будет равен $(T + 3)^2$.

Заметим, что при увеличении T на два на рисунке появляется дополнительная «рамка», состоящая из волн. Нетрудно понять, что для картины волн спустя T секунд количество клеток «рамки» равно $4T$.

Таким образом, спустя $T + 2$ секунды количество волн будет равно:

$$(T + 1)^2 + 4(T + 2) = T^2 + 2T + 1 + 4T + 8 = T^2 + 6T + 9 = (T + 3)^2.$$

Аналогичными рассуждениями можно доказать утверждение для четных значений T.

При решении следует пользоваться 64-битным типом данных, так как ответ может быть достаточно большим.

Сложность: $\mathcal{O}(1)$.

Задача 2. Фишки на поле

Для удобства сформулируем условие задачи несколько иначе: Алиса и Боб двигают свои фишки одновременно, требуется посчитать, сколько есть таких клеток, в которых одновременно оказалась как фишка Алиса, так и фишка Боба.

Пронумеруем строки доски сверху вниз числами от 0 до $N - 1$ и столбцы слева направо числами от 0 до $M - 1$. Запишем в клетке, находящейся на пересечении i-й строки и j-го столбца число $i + j$. Теперь заметим, что вся таблица разбилась на диагонали, на каждой из которых записаны одинаковые числа. Более того, любой ход перемещает фишку с диагонали с номером k на диагональ с номером $k + 1$. Это значит, что после выполнения k команд фишки Алисы и Боба окажутся на клетке, в которой записано число k.

Так как на каждой диагонали каждая из фишек побывает ровно один раз, то единственный случай, когда фишки могут пересечься на этой диагонали — это случай, при котором фишки окажутся в одной и той же клетке. Именно таким случаи мы и будем считать в новой формулировке задачи.

Для того, чтобы посчитать требуемое количество клеток, будем поддерживать четыре переменные: (i_A, j_A) — позиция фишки Алисы, а также (i_B, j_B) — позиция фишки Боба. Здесь i_A и i_B — это номера строк в таблице, а j_A и j_B — номера столбцов.

Изначально скажем, что ответ равен единице, так как обе фишки находятся на одной и той же стартовой клетке. Начнем обрабатывать перемещения фишке по одному. Во время каждого

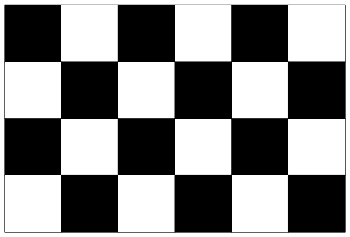
перемещения вычислим новые координаты фишки Алисы и новые координаты фишки Боба. Для этого нужно увеличить на единицу одну из координат фишки Алисы и фишки Боба, в зависимости от команд. После этого, если $i_A = i_B$, а также $j_A = j_B$, то нужно добавить к ответу единицу, так как мы нашли очередную подходящую клетку.

Сложность: $\mathcal{O}(N + M)$.

Задача 3. Ева красит доску

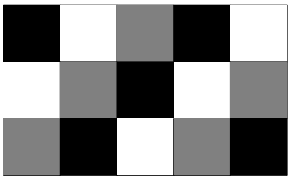
Для решения первой подзадачи можно было вручную решить каждый из возможных тестов. Для решения некоторых следующих подзадач можно было реализовать рекурсивный перебор с отсечениями. В зависимости от оптимальности отсечений, такое решение могло получать различное количество баллов.

Для решения задачи на полный балл нужно было придумать оптимальный способ покраски таблицы. Во-первых, очевидно, одного цвета не достаточно для покраски доски. Заметим, что в случае, если N и M четны, то можно воспользоваться так называемой *шахматной покраской*. Пример подобной покраски приведен ниже.

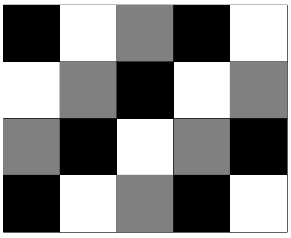


Заметим, что если N, либо M нечетно, то покрасить доску в два цвета невозможно, так как цвета крайних клеток в строках, либо в столбцах будут равны. Поэтому в этом случае потребуется хотя бы три цвета. Далее мы предложим способ покрасить любую доску в три цвета, поэтому большего количества цветов никогда не потребуется.

Попробуем покрасить доску следующим образом: первую строку покрасим в цвета $1, 2, 3, 1, 2, 3, \ldots$, вторую строку — в цвета $2, 3, 1, 2, 3, 1, \ldots$, третью строку — в цвета $3, 1, 2, 3, 1, 2, \ldots$, четвертую строку — в цвета $1, 2, 3, 1, 2, 3, \ldots$, и так далее. Пример подобной покраски приведен ниже.



Однако, иногда подобная покраска может быть неправильной. Например, если в таблице 7 столбцов, то каждая строка будет покрашена следующим образом: $c_1, c_2, c_3, c_1, c_2, c_3, c_1$. Видно, что цвета первой и последней клетки совпадают. Ниже приведен пример такой же проблемы, но уже не со строками, а со столбцами.

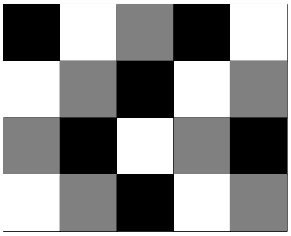


Заметим, что такая проблема со строками возникает в случае, если $M \equiv 1 \pmod 3$, а со столбцами — в случае, если $N \equiv 1 \pmod 3$. Данная проблема исправляется довольно легко:

- Если $M \equiv 1 \pmod 3$, заменим последний столбец на второй столбец.

- Если $N \equiv 1 \pmod 3$, заменим последнюю строку на вторую строку.

Нетрудно заметить, что в этом случае покраска станет правильной. Ниже приведена исправленная покраска таблицы, изображенной на предыдущем рисунке.



Сложность: $\mathcal{O}(N \cdot M)$.

Задача 4. Подпоследовательность и подмассив

Для решения задачи на частичный балл достаточно научиться проверять, является ли один массив подпоследовательностью другого массива.

Пусть мы хотим проверить, что массив a_2 является подпоследовательностью массива a_1. Для начала найдем в массиве a_1 элемент с минимальным индексом, равный первому элементу массива a_2. Обозначим индекс найденного элемента как p_1. Далее найдем в массиве a_1 элемент с минимальным индексом, большим p_1, который равен второму элементу массива a_2. Обозначим индекс найденного элемента как p_2. Продолжим по аналогии для остальных элементов. Если в какой-то момент времени не удается найти необходимый элемент в массиве a_1, то массив a_2 не является подпоследовательностью a_1. В противном случае подпоследовательность будет найдена.

Теперь для ответа на каждый запрос в явном виде выполним данный алгоритм. Сложность такого решения: $\mathcal{O}(Q \cdot N + \sum m_i)$. Конечно, при больших N и Q данное решение будет работать слишком долго.

Теперь рассмотрим решение данной задачи на полный балл. Очевидно, что нужно научиться выполнять операцию «найти в массиве a элемент с минимальным индексом, большим, чем p, равный x» быстрее, чем за линейный проход. Для начала запомним для каждого элемента x список таких индексов $i_1, i_2, \ldots, i_k$, что $a[i_j] = x$. Это можно сделать во время считывания массива a.

Теперь научимся выполнять требуемую операцию быстрее. Посмотрим в список индексов вхождения числа x в массив a. В нем требуется найти минимальный индекс, больший, чем p. Это можно выполнить при помощи бинарного поиска за $\mathcal{O}(\log N)$. В языке C++ это можно сделать при помощи встроенной функции `upper_bound`.

Таким образом, решение выглядит так: для каждого запроса пройдемся по элементам массива b_i, и каждый раз будем искать первое вхождение текущего элемента в массив a, индекс которого больше, чем p. Изначально $p = l_i - 1$. Если в какой-то момент времени p станет больше, чем r_i, то найти требуемую подпоследовательность невозможно.

Сложность: $\mathcal{O}(N + (\sum m_i) \log N)$.

Задача 5. Квантовые вычисления

Для решения задачи на частичный балл достаточно в явном виде запрограммировать то, что сказано в условии: во время первой фазы прибавить d_i ко всем компьютерам на отрезке $[l_i, r_i]$, после чего перебрать все отрезки компьютеров длины не более, чем K, и найти отрезок с суммой элементов, не превосходящей C. Такое решение работает за $\mathcal{O}(Q \cdot N \cdot K)$, так как всего отрезков длины, не превосходящей K, имеется $\mathcal{O}(N \cdot K)$.

Ключевая идея задачи заключается в том, что при прибавлении числа d_i к частотам некоторых компьютеров сумма на любом отрезке массива либо остается неизменной, либо увеличивается. Поэтому, если в какой-то момент времени в массиве не нашлось отрезка длины, не превосходящей K, с суммой, не большей, чем C, то такой отрезок больше никогда не найдется. Это значит, что можно перебрать номер первого эксперимента, когда подходящего отрезка не стало, при помощи бинарного поиска. Пусть бинарный поиск зафиксировал некоторый номер эксперимента X. Нужно выполнить

первые фазы всех экспериментов до X-го, включительно, после чего проверить, существует ли требуемый отрезок. Такое решение работает за $\mathcal{O}((Q \cdot N + N \cdot K) \log Q)$, где $\log Q$ — количество итераций бинарного поиска.

Теперь научимся быстро выполнять первые фазы экспериментов. Для этого нужно $\mathcal{O}(Q)$ раз выполнить операцию: прибавить d_i на отрезок $[l_i, r_i]$ массива. После всех операций нужно восстановить получившийся.

Заведем массив $delta[i]$ — какое значение должно быть прибавлено к $a[i]$. Изначально все $delta[i] = 0$. Пусть требуется прибавить число d_i на отрезок $[l_i, r_i]$. Вместо этого пока что прибавим число d_i к $delta[l_i]$ и вычтем d_i из $delta[r_i + 1]$. После выполнения этого действия для всех операций прибавления на отрезок вычислим на массиве $delta[i]$ префиксные суммы. Утверждается, что к элементу $a[i]$ должно быть прибавлено значение $delta[1]+delta[2]+\ldots+delta[i]$. Таким образом мы научились выполнять все операции прибавления на отрезок за $\mathcal{O}(Q+N)$ вместо $\mathcal{O}(Q \cdot N)$. Теперь все решение работает за $\mathcal{O}((Q + N \cdot K) \log Q)$.

Для решения задачи на полный балл осталось ускорить вторую часть решения — поиск отрезка, состоящего не более, чем из K элементов, и имеющего сумму, не превосходящую C. Для начала вычислим массив префиксных сумм на массиве a после применения всех прибавлений на отрезках. Обозначим этот массив как $pref[i]$. Для того, чтобы вычислить сумму на отрезке $[l, r]$ достаточно вычислить значение $pref[r] - pref[l - 1]$.

Переберем правую границу r отрезка, который мы будем рассматривать. Тогда мы можем получить суммы отрезков: $pref[r] - pref[r - 1], pref[r] - pref[r - 2], \ldots, pref[r] - pref[r - K]$. Выберем среди этих сумм минимальную. Если она не превосходит C, мы нашли нужный отрезок, иначе перейдем к рассмотрению другого значения r. Для того, чтобы выбрать минимальную сумму, нужно выбрать максимальное значение $pref[r - i]$ для всех i от 1 до K.

Воспользуемся структурой данных `set`, в которой будем хранить значения $pref[r - 1], pref[r - 2], \ldots, pref[r - K]$. Используя `set`, можно легко найти максимальный элемент, хранимый в нем. При переходе от r к $r + 1$ нужно удалить из множества элемент $pref[r - K]$ и добавить элемент $pref[r]$. Все операции со структурой данных `set` работают за $\mathcal{O}(\log K)$. Таким образом, полученное решение работает за $\mathcal{O}((Q + N \log K) \log Q)$. Данное решение проходит все группы тестов, кроме последней.

Для того, чтобы решить задачу на полный балл, необходимо еще раз ускорить вторую часть решения. Необходима структура данных, которая позволяет выполнять три операции:

- Добавить элемент в конец очереди.
- Удалить элемент из начала очереди.
- Найти максимальное число в очереди.

Выполнять все эти операции нужно за $\mathcal{O}(1)$. Такая структура данных существует, и называется *очередью с максимумом*. Данная структура данных может быть реализована на структуре данных `deque`. Подробнее вы можете почитать об этой структуре данных по ссылке: `http://shujkova.ru/sites/default/files/lec1.pdf`.

Сложность: $\mathcal{O}((Q + N) \log Q)$.