Задача A. Три и один

В задаче требовалось вычислить расстояния, которые пройдут три кота, перемещаясь между валунами, и сравнить их с расстояниями, которые нужно пройти четвёртому коту.

Обозначим координаты валунов (r_1, c_1), (r_2, c_2), (r_3, c_3), а координаты четвёртого кота (p, q).

Поскольку коты перемещаются только по дорожкам, то несложно вычислить:

- расстояние, которое пройдёт третий кот до **первого** валуна $s_1 = |r_1 - r_3| + |c_1 - c_3|$;
- расстояние, которое пройдёт первый кот до **второго** валуна $s_2 = |r_2 - r_1| + |c_2 - c_1|$;
- расстояние, которое пройдёт второй кот до **третьего** валуна $s_3 = |r_3 - r_2| + |c_3 - c_2|$.

Для четвёртого кота:

- расстояние, которое пройдёт четвёртый кот до **первого** валуна $t_1 = |p - r_1| + |q - c_1|$;
- расстояние, которое пройдёт четвёртый кот до **второго** валуна $t_2 = |p - r_2| + |q - c_2|$;
- расстояние, которое пройдёт четвёртый кот до **третьего** валуна $t_3 = |p - r_3| + |q - c_3|$.

(Напомним, что функция, вычисляющая абсолютную величину некоторого числа, в большинстве языков программирования называется *abs*.)

Теперь остается для каждого j сравнить s_j и t_j. Если окажется, что $t_j < s_j$, значит, четвёртый кот сможет занять этот валун, и нужно выводить номер этого валуна в ответе.

Разумеется, можно было воспользоваться списками / массивами для хранения расстояний.

Участники, для которых формирование ответа за один проход представляло сложности, могли выполнить сравнения дважды — сначала для подсчёта количества валунов, которые мог бы занять четвёртый кот, затем для вывода номеров этих валунов.

Задача B. Абсолютно относительно

Опишем один из возможных способов решения.

Перенумеруем направления числами от 0 до 3. Пусть N будет сопоставлено 0, E — 1, S — 2, W — 3. Это позволит вычислять разность между последовательными направлениями движения и, тем самым, определять направление в относительном пути.

Вычисления следует вести по модулю 4. Действительно, нумерацию мы провели «по часовой стрелке», поэтому за W при движении по часовой стрелке должно следовать N (иными словами, добавление числа, кратного 4, к любому направлению приведёт нас к этому же направлению после совершения нескольких полных оборотов). Поэтому, когда при вычислении разности будет получаться отрицательное число, номер направления из него можно получить прибавлением 4.

Заметим, что «с точки зрения» относительного пути платформа в текущий момент всегда ориентирована на север и меняет свое направление относительно этой ориентации. Следовательно, если разность между направлениями движения составит 0, то платформа продолжит движение в текущем направлении, если 1, это будет соответствовать повороту направо, если 2 — развороту в противоположном направлении, если 3 — повороту налево.

Рассмотрим пример из условия ($NWSEESNW$).

Поскольку изначально платформа ориентирована по направлению N, то разность направлений $0 - 0$ составит 0, поэтому первая команда в относительном пути также должна быть N.

Далее, в абсолютном пути выполнен поворот налево, разность направлений $3 - 0$ составит 3, и вторая команда в относительном пути также будет W.

Следующая команда в абсолютном пути S, но направление платформы в момент выдачи этой команды совпадает с W, поэтому разность направлений вычисляется как $2-3 = -1$. Чтобы получить положительное значение направления, следует добавить 4: $4 - 1 = 3$, поэтому следующая команда в относительном пути также будет W.

Аналогичным образом посчитаем разность для направления E в абсолютном пути: платформа ориентирована в направлении S, $1-2 = -1$, $-1+4 = 3$, что вновь даёт команду W в относительном пути.

Ещё одна команда E в абсолютном пути с точки зрения пути относительного выглядит как сохранение направления. После выполнения предыдущей команды платформа ориентирована в направлении E, формальный подсчёт даёт $1 - 1 = 0$ и команду N в ответ.

Затем команда S в абсолютном пути приводит к $2 - 1 = 1$ команде E в пути относительном.

Команда N должна развернуть платформу в противоположном направлении, $0 - 2 = -2$, после добавления 4 получим $4 - 2 = 2$ и команду S в относительном пути.

Наконец, последняя команда W повторяет выкладки, сделанные во втором шаге, что позволяет получить завершающую относительны путь команду W.

Действуя таким образом можно преобразовать исходную строку абсолютного пути в относительный.

Разумеется, можно было рассмотреть всевозможные комбинации пар команд (16) и получить для каждой пары соответствующую команду в относительном пути. Если в маршруте допускалось ровно два возможных направления, то реализовать такое для каких-то пар было несколько проще.

Задача C. Вывоз мусора

В этой задаче нужно было выполнить симуляцию описанного процесса.

Сохраним координаты выездов с аллеи в списке (массиве или иной линейной структуре данных) c, а координаты мешков с мусором — в списке (массиве или иной линейной структуре данных) b.

Поскольку и координаты выездов, и координаты мешков с мусором упорядочены по неубыванию, то мы можем быть уверены в следующем. Пусть, забрав в ковш некоторый мешок b_i ($i \mod k = 0$), погрузчик направляется к ближайшему выезду. Пусть ближайшим выездом является выезд c_j. Тогда для мешка b_{i+k} ближайшим выездом будет выезд с индексом, не меньшим j. Поэтому можно двигаться по списку координат выездов и списку координат мешков с мусором «параллельными курсами», никогда не возвращаясь назад (это один из вариантов техники двух указателей).

Ещё один важный момент состоит в том, что нам следует фиксировать не только абсолютное значение расстояния от мешка с мусором до выезда, но и его знак. Будем полагать, что знак отрицательный, когда координата выезда меньше координаты мешка, и положительный в противном случае. Несложно понять, что в общем случае надо сравнивать расстояния от мешка до двух выездов — «последнего отрицательного» и «первого положительного». Поэтому если текущий рассматриваемый выезд для некоторого мешка «отрицательный», нужно перебирать следующие выезды, пока не встретится «положительный». А если текущий рассматриваемый выезд для некоторого мешка «положительный», то ничего перебирать не нужно — лучший вариант уже найден. Разумеется, если в какой-то момент выезды закончились, то последний выезд будет использоваться для всех оставшихся мешков.

Для организации такого перебора можно использовать цикл с предусловием; при этом индексы списков следует описать до цикла, а в самом цикле изменять их согласно описанным выше условиям (фиксируя один из индексов и изменяя другой).

Асимптотика описанного алгоритма составляет $O(n + m)$.

Ограничения в первой подзадаче позволяли решать задачу алгоритмом с асимптотикой $O((n/k) \cdot m)$ (перебирая все выезды в поисках подходящего для каждого k-го мешка с мусором).

Вторая подзадача предполагала некоторую оптимизацию для «большого» количества мешков с мусором.

Аккуратно написанный бинпоиск также позволял получить полный балл.

Задача D. Шторы

Общее замечание.

Для всех описанных ниже решений подразумевается, что случай крючков 1 и n обрабатывается отдельно до любых последующих вычислений.

Решение для $n \leqslant 10$

Для каждого запроса проэмулируем пошагово процесс. Если в процессе обработки нашли крючок из запроса — останавливаем эмуляцию и выводим номер текущего шага.

Для эмуляции будем поддерживать массив логических значений «использован ли крючок i».

Перед началом эмуляции помечаем крючки 1 и n как использованные.

Рассмотрим процесс поиска левого наибольшего отрезка.

Используем следующие величины:

- Длина $MaxLen$ и начало $MaxStart$ левого наибольшего среди уже найденных отрезков — изначально $MaxLen = 0$, $MaxStart = -1$.
- Длина $CurLen$ и начало $CurStart$ текущего рассматриваемого отрезка — изначально $CurLen = 0$, $CurStart = -1$.

Сам поиск отрезка:

- Обрабатываем крючки слева направо (от 1 до n включительно).
- Если текущий крючок p уже использован, то:
  - Если $MaxLen < CurLen$, то выполняем изменения $MaxLen = CurLen$, $MaxStart = CurStart$;
  - Выполняем изменения $CurLen = 0$, $CurStart = -1$.
- Если текущий крючок p ещё не использован, то:
  - Если $CurStart = -1$, то выполняем изменение $CurStart = p$;
  - Выполняем изменение $CurLen = CurLen + 1$.

После выбора нужного отрезка помечаем центральный крючок (крючки) как использованные. Если крючок из запроса не использован — переходим к следующему шагу.

Итого получается $O(n)$ действий на одном этапе, всего этапов $O(n)$, получаем $O(n^2)$ на эмуляцию на каждом запросе — всего $O(q \cdot n^2)$.

Решение для $n \leqslant 10^3$

Заметим, что можно проэмулировать процесс один раз и запомнить для всех крючков, на каком шаге они использованы.

После этого будем отвечать на запросы за $O(1)$, доставая информацию из вычисленного массива.

Итого получается $O(n)$ действий на одном этапе, всего этапов $O(n)$, получаем $O(n^2)$ на эмуляцию и $O(q)$ на последующие ответы на запросы — всего $O(n^2 + q)$.

Решение для $n \leqslant 3 \cdot 10^5$

Заметим, что центральные крючки выбираются таким образом, что отрезок всегда делится на две части **равной длины**.

Поэтому существует всего $O(\log n)$ возможных длин отрезков крючков:

- На первом этапе один отрезок длины $n - 2$;
- Затем образуются два отрезка длины $(n - 2)/2$;
- Из этих отрезков образуются четыре отрезка длины $(n - 2)/4$;
- И так далее до отрезков длины 2 или 1.

Это позволяет оптимизировать поиск левого из наибольших неиспользованных отрезков для нашей эмуляции.

Будем поддерживать текущие необработанные отрезки в **очереди** (`https://ru.wikipedia.org/wiki/Очередь_(программирование)`) — каждый отрезок задаётся парой (начало; конец).

На каждой итерации будем:

- Доставать отрезок из начала очереди (он является левым наибольшим на данном этапе);
- Помечать его центральные крючки и запоминать для них номер шага;
- Если оставшиеся части отрезка имеют ненулевую длину, то кладем их в конец очереди (сначала левый, потом правый).

Итого на эмуляцию требуется $O(1)$ действий на каждом из $O(n)$ этапов — всего $O(n+q)$ на предпросчет и последующие запросы.

Решение для $t_i \leqslant 3 \cdot 10^5$

Проэмулируем наш процесс только на $\max t_i$ шагов.

Так как номера крючков могут быть очень большими, то вместо словаря для запоминания ответов будем использовать **словарь** (`https://ru.wikipedia.org/wiki/Ассоциативный_массив`).

Таким образом всё решение использует $O(\max t_i + q)$ действий.

Решение для полных ограничений

Разобьем все отрезки по их длинам на $O(\log n)$ слоёв — k-й слой содержит отрезки с длинами $(n-2)/2^k$.

Пусть интересующий нас крючок p будет использован как центр отрезка на слое k.

Итоговый номер шага для крючка p складывается из двух частей:

1. Суммарное количество отрезков со строго большими длинами, чем отрезок с центром p — это $2^0 + 2^1 + \ldots + 2^{k-1}$.
2. Количество отрезков длины $(n-2)/2^k$, стоящие строго левее отрезка с центром p.

Также обратим внимание, что на каждом слое от 0 до k существует **ровно один отрезок**, содержащий крючок p.

Научимся вычислять номер шага для заданного крючка p за $O(\log n)$ действий (по количеству слоёв).

Будем поддерживать следующие величины:

- Номер текущего слоя k — изначально $k = 0$.
- Границы текущего рассматриваемого отрезка $[L; R]$ — изначально $L = 2$, $R = n - 1$.
- Номер отрезка на текущем слое x (в 0-индексации) — изначально $x = 0$.
- Количество обработанных отрезков на предыдущих слоях b — изначально $b = 0$.

Процесс вычисления будет следующим:

- Пусть m_L и m_R — центры отрезка $[L; R]$.
- Если $p = m_L$ или $p = m_R$, то ответом для крючка является $b + x + 2$, останавливаем процесс.
- Иначе увеличиваем b на 2^k, k увеличиваем на 1 и:
  - Если $p < m_L$, то выполняем изменения $R = m_L - 1$; $x = 2 \cdot x$;
  - Иначе выполняем изменения $L = m_R + 1$; $x = 2 \cdot x + 1$.

Итого на один запрос требуется $O(\log n)$ действий — всего $O(q \cdot \log n)$.