Задача A. Холодный осенний вечер

Суммарное количество сторон деталей, на которых есть отверстия, равно количеству пар соседних по стороне пазлов. В каждом из N рядов есть $M-1$ пара соседних пазлов, и в каждом из M столбцов есть $N-1$ пара соседних пазлов. Всего получаем $N \cdot (M-1) + M \cdot (N-1) = 2 \cdot N \cdot M - N - M$ отверстий.

Задача B. Лучший выбор

Заметим, что если i — это номер младшего нулевого бита X, то мы не можем ни из какой копилки получить 2^i бубликов, соответственно MEX не может стать больше 2^i. При этом для любого $a < 2^i$ верно, что $a \& X = a$. Соответственно, ответ на задачу равен $\min(N, 2^i)$.

Задача C. Повторное прохождение

Если для какой-то клетки (x, y) ее нижний и правый соседи будут иметь одинаковые значения, то найдутся хотя бы два пути с одинаковыми следами. Иначе, если такой клетки не существует, то все пути будут иметь различные следы.

Чтобы это доказать, рассмотрим первый момент, когда появятся одинаковые следы. Заметим, что до последнего хода эти следы должны были быть либо одним и тем же путем (и тогда клетки-соседи все-таки оказались равны), либо у нас должны были быть два равных следа еще на прошлом шаге (что противоречит исходному предположению).

Теперь, нужно найти количество матриц $N \times M$, в которых каждая побочная диагональ не содержит двух соседних равных чисел. Можно сказать, что для нижней левой клетки каждой диагонали можно свободно выбирать любое число от 1 до K, а во всех следующих клетках появляется одно запрещенное число. Таким образом, ответ на задачу равен $K^{N+M-1} \cdot (K-1)^{(N-1)(M-1)}$.

Чтобы посчитать эту формулу нужно использовать бинарное возведение в степень по модулю.

Задача D. Вадим и судоку: набнер

Заметим, что если подотрезок $[l, r]$ является набнером, то подотрезок $[l+1, r]$ так же является набнером. Благодаря этому, в данной задаче можно использовать метод двух указателей.

В процессе перебора нужно поддерживать множество всех использованных чисел, а также при добавлении нового числа x проверять, что в этом множестве нет чисел $x-1$, x и $x+1$.

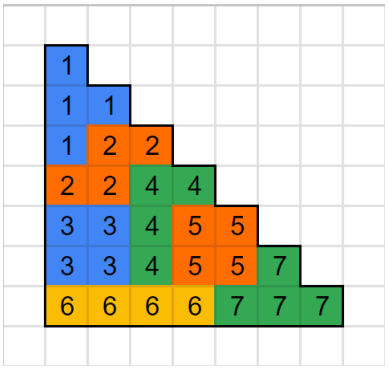
Задача E. Вадим и судоку: галактики

Отразим матрицу относительно центра галактики, строим граф в котором доступны только те клетки, которые доступны и в изначальной и в отраженной матрице. Теперь запускаем bfs из центра галактики, все посещённые вершины и есть максимальная галактика.

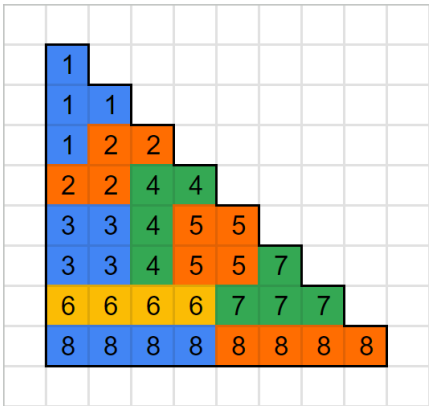
Задача F. Игнат и судоку: киллер-регионы

Для начала поймем для каких значений N ответ точно не существует. В треугольнике высоты N количество клеток равно $\frac{N(N+1)}{2}$. Это число делится на 4, если $N(N+1)$ делится на 8. То есть N имеет остаток 0 или 7 по модулю 8. Если остаток будет другим, то ответ «*NO*».

Решим задачу для $N = 7$:



Модифицируем это решение для $N = 8$:



Далее достаточно расположить блоки с $N = 7$ и $N = 8$ на границе треугольника, а всю внутренность можно заполнить полосками 1×4.

Задача G. Ненавижу шахматы

Изначально каждая клетка это отдельная компонента связности. За все запросы будет затронуто не более $5 \cdot Q$ клеток, так как мы можем учитывать только те клетки, которые будут в запросах, и соседние с ними по стороне. Создадим систему непересекающихся множеств для этих клеток и будем обновлять после каждого запроса, который красит белую клетку в чёрную. Изначально компонент связности $N \cdot N$. При каждом объединении множеств в СНМ будем уменьшать количество компонент на одну.

Задача H. Бесконечная сила

Математическое ожидание урона от одной атаки равно $E = \sum_{i=1}^{n} \frac{1+p_i}{2}$, так как равно сумме матожиданий по каждому отдельному зачарованию. Вероятность повторить атаку равна $p = m \cdot \prod_{i=1}^{n} \frac{1}{p_i}$, где m — это минимальное значение в массиве p.

Тогда матожидание урона серии атак равно $E + pE + p^2E + \dots$. Это можно записать как $E \cdot (1 + p + p^2 + \dots)$, что равно $\frac{E}{1-p}$.

Задача I. Работа — не волк, волк — это ходить

Чтобы проверить, является ли t периодом S, достаточно проверить, что $S_{|t|+1\dots|S|} = S_{1\dots|S|-|t|}$, сделать это можно префиксными хешами. Чтобы быстро заменять подстроку на строку из словаря и пересчитывать хеши, воспользуемся ДО с массовым присвоением на отрезке и значением хеша в вершине. Пусть t_1 в t_2 — периоды S, заметим, что префикс длины $gcd(|t_1|, |t_2|)$ также является периодом S, следовательно, мы можем найти минимальный период через факторизацию. Итоговое время работы $O(q \cdot log^2(|S|))$

Задача J. Глубокое понимание

Решим задачу с помощью динамического программирования. Пусть $dp_{i,j}$ будет равно числу упорядоченных корневых деревьев с глубиной j, состоящих из i вершин.

Каждое дерево, состоящее хотя бы из двух вершин, можно однозначно разбить на две части — поддерево самого правого ребенка корня и все остальное дерево без этой части. Тогда в пересчете можно перебрать размер правого поддерева и его глубину, и тогда получается следующая формула пересчета: $dp_{i,j} = \sum_{x=1}^{i-1} \sum_{y=1}^{j-2} dp_{i-x,j} + \sum_{x=1}^{i-1} \sum_{z=0}^{j-1} dp_{i-x,z}$.

Эту формулу можно оптимизировать с использованием префиксных сумм. Тогда решение будет работать за $O(N^3)$.

Задача K. Фирменный вопрос

Используем алгоритм Мо. Будем решать в пределах одного блока. Если запрос целиком лежит внутри, то посчитаем его так: пройдёмся от l_i до r_i и будем добавлять значения в set, при встрече нового будем искать ближайшие к $-x$ в set-е, обновлять по-необходимости. Работает за $O(\sqrt{N} \cdot log(N))$ на запрос.

Теперь рассматриваем все остальные запросы. Они разбиваются правой границей блока на два подотрезка, значит есть три случая: оптимальный ответ лежит в левом подотрезке (внутри блока), оптимальный ответ лежит в правом подотрезке (вне блока), оптимальный ответ лежит в различных подотрезках. Первые два решаются аналогично — мы идём от правой границы до начала блока и от правой границы до конца массива и ищем оптимальный ответ с помощью *set*-а, если попадается граница запроса, то обновляем для него ответ. Это суммарно работает за $O(\sqrt{N} \cdot log(N))$ для первого типа и $O(N \cdot log(N))$ для второго. Остался третий, для таких заведём *set* для правой половины чисел, по порядку будем добавлять их, если встречается правая граница запроса, то проверяем все числа (ищем ближайшие к $-x$ в *set*-е) от правой границы блока до левой границы запроса. Тогда на добавление нового числа справа суммарная сложность будет $O(N \cdot log(N))$, а на проверку одного запроса — $O(\sqrt{N} \cdot log(N))$.

Общая сложность выполнения алгоритма будет равна $O((N + Q) \cdot \sqrt{N} \cdot log(N))$.