Задача 329. Longest Increasing Path in a Matrix

Условие задачи

Дана матрица размера $m \times n$ с целыми числами. Требуется найти длину самого длинного возрастающего пути в матрице. Из каждой клетки можно двигаться в четырех направлениях: влево, вправо, вверх или вниз. Движение по диагонали или выход за границы матрицы запрещены (то есть нельзя двигаться по кругу). Путь называется возрастающим, если значения в последовательных клетках строго возрастают.

Пример 1:

$$\begin{bmatrix} 9 & 9 & 4 \\ 6 & 6 & 8 \\ 2 & 1 & 1 \end{bmatrix}$$

Входные данные: matrix = [[9,9,4],[6,6,8],[2,1,1]]
Выходные данные: 4
Пояснение: Самый длинный возрастающий путь [1, 2, 6, 9]

Пример 2:

$$\begin{bmatrix} 3 & 4 & 5 \\ 3 & 2 & 6 \\ 2 & 2 & 1 \end{bmatrix}$$

Входные данные: matrix = [[3,4,5],[3,2,6],[2,2,1]]
Выходные данные: 4
Пояснение: Самый длинный возрастающий путь [3, 4, 5, 6]. Перемещение по диагонали запрещено.

Пример 3:

Входные данные: matrix = [[1]]
Выходные данные: 1

Ограничения:

- $m ==$ matrix.length

- $n ==$ matrix[i].length
- $1 \leq m, n \leq 200$
- $0 \leq \text{matrix}[i][j] \leq 2^{31} - 1$

Анализ задачи

Путь может начинаться в любой клетке, поэтому необходимо определить максимальную длину возрастающего пути для каждой клетки матрицы.

Рассмотрим матрицу как ориентированный граф: каждая клетка (i, j) является вершиной, а ориентированное ребро проводится из (i, j) в соседнюю клетку (x, y), если клетки соседние по стороне и

$$matrix[x][y] > matrix[i][j].$$

Так как переход осуществляется только к большему значению, в полученном графе невозможны циклы. Следовательно, граф является ориентированным ациклическим графом.

Динамическое программирование

Обозначим через

$$dp[i][j]$$

длину самого длинного возрастающего пути, начинающегося в клетке (i, j).

Тогда выполняется рекуррентное соотношение:

$$dp[i][j] = \begin{cases} 1 + \max dp[x][y], & \text{если существует сосед } (x, y), \\ & \text{такой что } matrix[x][y] > matrix[i][j], \\ 1, & \text{если таких соседей нет.} \end{cases}$$

Таким образом, значение в клетке определяется через оптимальные значения в соседних клетках с большими числами.

Описание алгоритма

1. Инициализация

Создаётся вспомогательная матрица dp размера $m \times n$, заполненная нулями. Ноль означает, что значение ещё не вычислено.

2. Поиск в глубину (DFS) с мемоизацией
Для каждой клетки (i, j) вызывается рекурсивная функция:

- если $dp[i][j] \neq 0$, возвращается сохранённое значение;
- иначе устанавливается $max\_len = 1$;
- перебираются четыре направления;
- если сосед находится в пределах матрицы и его значение строго больше текущего, выполняется рекурсивный вызов;
- результат сохраняется в $dp[i][j]$.

3. Получение ответа
Ответом является

$$\max_{0 \leq i < m,\ 0 \leq j < n} dp[i][j].$$

Корректность алгоритма

Лемма 1 (Отсутствие циклов)
В построенном графе циклы невозможны.

Доказательство: Каждое ребро направлено от меньшего значения к большему. Предположим, что существует цикл. Тогда значения вдоль цикла должны строго возрастать, но при возвращении в исходную вершину значение должно быть одновременно больше и равно самому себе, что невозможно. Следовательно, граф ацикличен.

Лемма 2 (Оптимальная подструктура)
Если путь из клетки (i, j) является самым длинным возрастающим, то его продолжение из следующей клетки также является максимальным.

Доказательство: Пусть оптимальный путь имеет вид

$$(i, j) \to (x_1, y_1) \to \dots$$

Если подпуть, начинающийся в (x_1, y_1), не является максимальным, то его можно заменить более длинным, что увеличит длину исходного пути. Получаем противоречие. Следовательно, выполняется рекуррентная формула.

Лемма 3 (Корректность мемоизации)
Каждое значение $dp[i][j]$ вычисляется не более одного раза.

Доказательство: Так как граф ацикличен, рекурсивные вызовы всегда переходят к вершинам с большими значениями. После вычисления значение сохраняется в массиве dp, и при повторном обращении пересчёт не производится. Следовательно, каждая подзадача решается единственный раз.

Следствие

Рекуррентная формула для $dp[i][j]$ корректно определяет длину самого длинного возрастающего пути, так как каждая клетка зависит только от клеток с большими значениями, а построенный граф является ацикличным. Следовательно, вычисления могут быть выполнены без противоречий и зацикливания, и найденное значение является оптимальным.

Из лемм следует корректность алгоритма.

Разбор примера

Рассмотрим матрицу:

$$\begin{bmatrix} 9 & 9 & 4 \\ 6 & 6 & 8 \\ 2 & 1 & 1 \end{bmatrix}$$

Один из самых длинных возрастающих путей:

$$1 \to 2 \to 6 \to 9$$

Вычисления происходят следующим образом:

$$dp(9) = 1,$$

$$dp(6) = 1 + dp(9) = 2,$$

$$dp(2) = 1 + dp(6) = 3,$$

$$dp(1) = 1 + dp(2) = 4.$$

Алгоритм автоматически вычисляет значения в корректном порядке за счёт рекурсивных вызовов и мемоизации.

Анализ сложности

Временная сложность

Каждая клетка матрицы соответствует вершине графа. Так как благодаря мемоизации значение $dp[i][j]$ вычисляется не более одного раза, каждая вершина обрабатывается единственный раз.

Для каждой клетки проверяются не более четырёх соседей, что занимает константное время.

Следовательно, общее число операций пропорционально числу клеток матрицы:

$$O(m \cdot n).$$

Пространственная сложность

Используется вспомогательная матрица dp размера $m \times n$, что требует

$$O(m \cdot n)$$

дополнительной памяти.

В худшем случае глубина рекурсии может достигать $O(m \cdot n)$, если возрастающий путь проходит через все клетки.

Вывод

Задача решается с использованием поиска в глубину с мемоизацией. Представление матрицы в виде ориентированного ациклического графа позволяет строго обосновать корректность алгоритма. Полученное решение эффективно по времени и памяти и удовлетворяет ограничениям задачи.