ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу

Мирасов Константин

Академическая группа J3112

Принято

Ассистент, Дунаев Максим

Санкт-Петербург

2024

Введение

Цель:
Разработать алгоритм для нахождения площади максимального прямоугольника, содержащего только единицы, в бинарной матрице, заданной символами '0' и '1'.

Задачи:

1. Определить подход, обеспечивающий эффективное решение задачи.
2. Реализовать алгоритм на языке C++.
3. Выполнить анализ сложности по времени и памяти.
4. Обосновать выбор метода динамического программирования.

Теоретическая подготовка

Постановка задачи:
Дана бинарная матрица размером rows×cols, содержащая символы '0' и '1'. Необходимо найти максимальную площадь прямоугольника, заполненного только '1'.

Ключевые подходы:

Подход наивного перебора:

- Проверить все возможные прямоугольники, подсчитав количество единиц.
- Сложность такого подхода O((rows·cols)^2), что не подходит для больших матриц.

Динамическое программирование:

- Воспользоваться свойством подзадач: высоты столбцов для текущей строки зависят от предыдущей строки.
- Сводим задачу к поиску максимальной площади в гистограмме для каждой строки.
- Временная сложность сокращается до O(rows·cols).

Почему динамическое программирование?
Динамическое программирование необходимо, так как:

Разбиение задачи на подзадачи:

- Задача поиска максимального прямоугольника для всей матрицы делится на поиск максимальной площади в гистограмме для каждой строки.
- Высоты столбцов обновляются динамически: если текущая ячейка равна '1', высота увеличивается, иначе сбрасывается в 0.

Избежание повторных вычислений:

- Вместо повторного анализа предыдущих строк используется массив высот, который накапливает информацию.

Эффективность:

- Сложность решения O(rows·cols), что существенно быстрее наивного подхода.

Практическая часть

Алгоритм:

1 — Инициализировать массив heights для хранения высот колонок.

2 — Для каждой строки:

- Обновить массив heights в зависимости от значений текущей строки.
- Найти максимальную площадь прямоугольника, используя алгоритм поиска в гистограмме с помощью стэка.

3 — Вернуть максимальную площадь.

Код:

```cpp
class Solution {
public:
    int maximalRectangle(std::vector<std::vector<char>>& matrix) {
        if (matrix.empty()) return 0; // O(1)

        int rows = matrix.size(); // O(1)
        int cols = matrix[0].size(); // O(1)
        std::vector<int> heights(cols, 0); // O(cols)
        int maxArea = 0; // O(1)

        // Внешний цикл проходит по всем строкам матрицы - O(rows)
        for (int i = 0; i < rows; ++i) {
            // Внутренний цикл для обновления высот - O(cols)
            for (int j = 0; j < cols; ++j) {
                // Обновляем массив высот: O(1) на итерацию
                heights[j] = (matrix[i][j] == '1') ? heights[j] + 1 : 0;
            }

            // Вычисляем максимальную площадь прямоугольника для текущей строки (гистограммы): O(cols)
            maxArea = std::max(maxArea, largestRectangleArea(heights));
        }

        return maxArea; // O(1)
    }

private:
    int largestRectangleArea(std::vector<int>& heights) {
        heights.push_back(0); // Добавляем фиктивное значение: O(1)
        std::stack<int> stack; // O(1)
        int maxArea = 0; // O(1)

        // Проходим по массиву высот, включая фиктивный элемент: O(n), где n — размер heights
        for (int i = 0; i < heights.size(); ++i) {
            // Обрабатываем стек, пока текущая высота меньше верхнего элемента стека: O(n) амортизированно
            while (!stack.empty() && heights[i] < heights[stack.top()]) {
                int height = heights[stack.top()]; // O(1)
                stack.pop(); // O(1)
                int width = stack.empty() ? i : i - stack.top() - 1; // O(1)
                maxArea = std::max(maxArea, height * width); // O(1)
            }
            stack.push(i); // O(1)
        }

        heights.pop_back(); // Удаляем фиктивное значение: O(1)
        return maxArea; // O(1)
    }
};
```

Анализ сложности:

1 — **Временная сложность**:

Обновление массива высот для каждой строки: O(rows·cols).

Нахождение максимальной площади в гистограмме: O(rows·cols).

Итоговая сложность: O(rows·cols).

2 — Память:

Массив высот heights: O(cols).

Стэк для вычислений в гистограмме: O(cols)).

Итоговое использование памяти: O(cols).

Вывод

Оптимизация подпроблем:

Задача для всей матрицы сводится к подзадачам — вычислению высот колонок и нахождению площади для каждой строки. Эти подзадачи зависят от предыдущих строк, что делает задачу идеальной для подхода динамического программирования.

Избежание повторных вычислений:

Вместо того чтобы анализировать все возможные прямоугольники, динамическое программирование позволяет накопить результаты предыдущих вычислений (высоты колонок) и быстро переходить к следующей строке.

Эффективность:

Динамическое программирование сводит задачу с квадратичной сложности O((rows·cols)^2) в случае наивного решения) к линейной по числу элементов матрицы O(rows·cols).