ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 7

«Жадные алгоритмы»

Выполнил работу

Ковров Евгений

Академическая группа №J3110

Принято

Должность, звание Вершинин Владислав

Санкт-Петербург

2024

1. Введение

Цель: реализовать алгоритм поиска максимального прямоугольника из “1” в матрице, заполненной “0” и “1”.

Задачи:

1. Постановка задачи
2. Алгоритм
3. Реализация
4. Тестирование
5. Оптимизация

Теоретическая подготовка

Основной подход, использованный для решения задачи – динамическое программирование. Данный подход удобен, так как решение задачи предполагает наличие большого количества схожих подзадач.

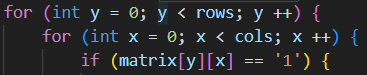
1. Реализация

Постановка задачи: Дана двоичная матрица (rows x cols), заполненная 0 и 1. Необходимо найти наибольший прямоугольник, содержащий только 1 и вернуть его площадь.

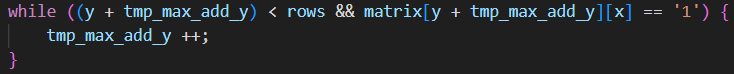
Алгоритм**:**

1. Последовательный перебор всех “ячеек” в качестве левой верхней для искомого прямоугольника.
2. Для каждой такой ячейки перебираем все возможные значения длины вертикальной стороны прямоугольника.
3. Для каждой длины вертикальной стороны определяем максимальную горизонтальную сторону.
4. Вычисляем текущую площадь и сравниваем с максимальной

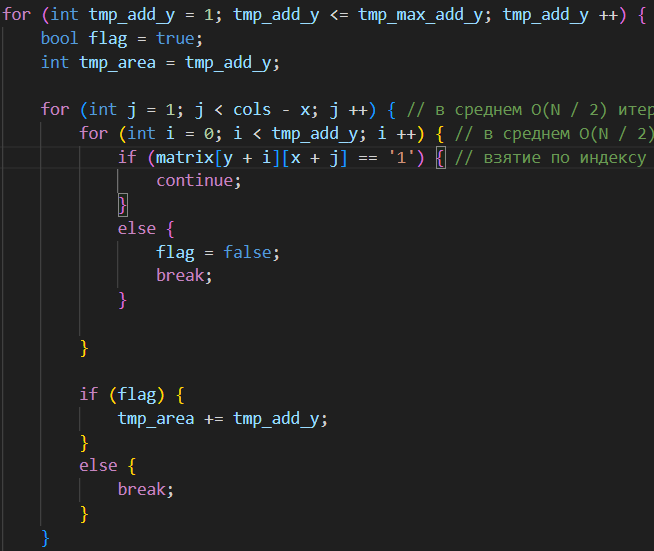
Реализация:



Изображение №1 – Перебор всех точек в качестве вершины



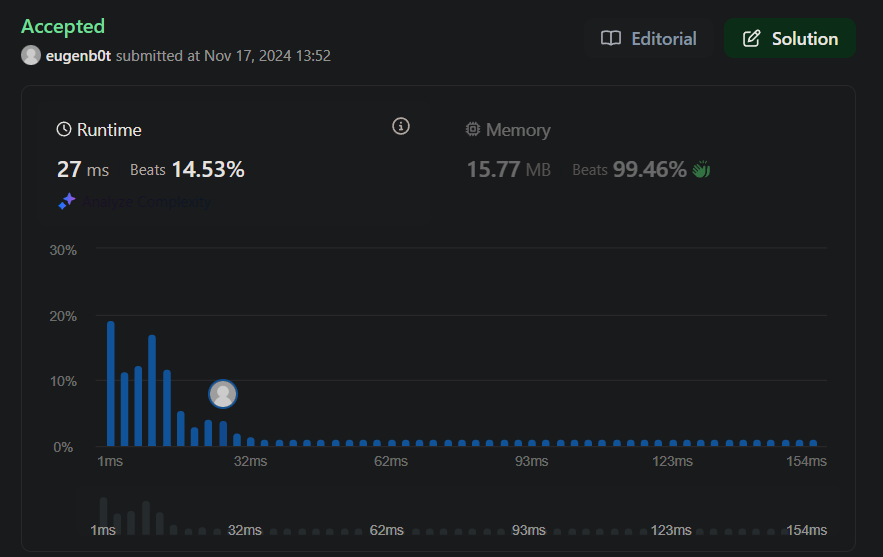
Изображение №2 – Определение максимальной возможной “глубины”

  
Изображение № 3 – Последовательно добавляем лежащие справа столбцы

Экспериментальная часть

Тестирование:

Так как задача была взята с leetcode, она была протестирована тестами на платформе (изображение № 4).



Изображение № 4 – Отчёт leetcode

Подсчёт памяти:

Дополнительная память для сложных структур не используется. Для каждого прямоугольника сохраняется площадь в формате int. Поэтому память можно оценить как O(N ^ 3).

Подсчёт асимптотики:

Для удобства будем считать прямоугольную матрицу квадратной размером N x N.



Изображение № 5 – Перебор всех ячеек матрицы – O(N ^ 2)



Изображение № 6 – Перебор всех возможных “глубин” – в среднем O(N / 2)



Изображение № 7 – Перебор всех столбцов текущей глубины – в среднем

O((N ^ 2) / 4)

Итоговая теоретическая сложность – O(N ^ 5)

В таблице 1 приведены экспериментально полученные значения времени работы алгоритма для различных размеров исходной матрицы.

|  |  |
| --- | --- |
| Размер | Время |
| 1 | 3.00E-07 |
| 2 | 6.00E-07 |
| 3 | 1.00E-06 |
| 4 | 1.20E-06 |
| 5 | 1.40E-06 |
| 6 | 2.90E-06 |
| 7 | 3.20E-06 |
| 8 | 4.30E-06 |
| 9 | 7.60E-06 |
| 10 | 5.70E-06 |
| 11 | 6.00E-06 |
| 12 | 6.50E-06 |
| 13 | 8.80E-06 |
| 14 | 1.20E-05 |
| 15 | 1.58E-05 |
| 16 | 2.09E-05 |
| 17 | 1.55E-05 |
| 18 | 1.64E-05 |
| 19 | 2.10E-05 |
| 20 | 2.42E-05 |
| 21 | 2.24E-05 |
| 22 | 2.78E-05 |
| 23 | 2.82E-05 |
| 24 | 3.28E-05 |
| 25 | 3.28E-05 |
| 26 | 3.68E-05 |
| 27 | 3.75E-05 |
| 28 | 4.09E-05 |
| 29 | 4.98E-05 |
| 30 | 5.32E-05 |
| 31 | 5.01E-05 |
| 32 | 4.80E-05 |
| 33 | 6.63E-05 |
| 34 | 6.21E-05 |
| 35 | 7.20E-05 |
| 36 | 7.45E-05 |
| 37 | 7.07E-05 |
| 38 | 9.85E-05 |
| 39 | 8.16E-05 |
| 40 | 8.58E-05 |
| 41 | 8.95E-05 |
| 42 | 9.52E-05 |
| 43 | 9.27E-05 |
| 44 | 9.82E-05 |
| 45 | 0.000112 |
| 46 | 0.000114 |
| 47 | 0.000128 |
| 48 | 0.000142 |
| 49 | 0.00012 |
| 50 | 0.000133 |
| 51 | 0.00014 |
| 52 | 0.000152 |
| 53 | 0.000149 |
| 54 | 0.000161 |
| 55 | 0.000161 |
| 56 | 0.000164 |
| 57 | 0.000167 |
| 58 | 0.000164 |
| 59 | 0.000179 |
| 60 | 0.000189 |
| 61 | 0.000201 |
| 62 | 0.000214 |
| 63 | 0.000206 |
| 64 | 0.000209 |
| 65 | 0.000238 |
| 66 | 0.000229 |
| 67 | 0.000255 |
| 68 | 0.000224 |
| 69 | 0.000248 |
| 70 | 0.00026 |
| 71 | 0.000254 |
| 72 | 0.000268 |
| 73 | 0.000272 |
| 74 | 0.000289 |
| 75 | 0.000288 |
| 76 | 0.000288 |
| 77 | 0.000317 |
| 78 | 0.000329 |
| 79 | 0.000337 |
| 80 | 0.000334 |
| 81 | 0.00034 |
| 82 | 0.000357 |
| 83 | 0.000367 |
| 84 | 0.000358 |
| 85 | 0.000386 |
| 86 | 0.000381 |
| 87 | 0.000396 |
| 88 | 0.000407 |
| 89 | 0.000398 |
| 90 | 0.000428 |
| 91 | 0.000417 |
| 92 | 0.000426 |
| 93 | 0.000447 |
| 94 | 0.000443 |
| 95 | 0.00046 |
| 96 | 0.000468 |
| 97 | 0.000501 |
| 98 | 0.000502 |
| 99 | 0.000524 |

Таблица № 1 – Зависимость времени от размера

На основе данных из таблицы. Был построен график (Изображение № 7).

Изображение № 8– График реальной асимптотики алгоритма

Такие сильные расхождения в сложности могут быть вызваны маленькими значениями времени. Так как «честная» теоретическая асимптотика равна O((N ^ 5) / 8), график может вести себя так на маленьких значениях. Также в среднем для половины ячеек матрицы (равных “0”) алгоритм поиска не запускается в принципе.

Оптимизация

Было добавлено моментальное завершение алгоритма в случае, если вся матрица состоит из “1”.



Изображение № 8 – Преждевременный выход из функции

1. Заключение

В ходе выполнения работы мною был реализован алгоритм поиска наибольшего прямоугольника, состоящего из “1”, в бинарной матрице. Цель работы была достигнута путём тестирования на матрицах различного размера с разным количеством “0” и “1”. Полученные результаты не совпадают с теоретическими оценками сложности алгоритма, однако тому есть логическое обоснование.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения асимптотики и добавление большего количества преждевременных выходов в качестве исключений.

sПРИЛОЖЕНИЕ А

Листинг кода файла max\_rectangle.cpp

#include <iostream>

#include <vector>

#include <algorithm>

#include <fstream>

#include <sstream>

#include <chrono>

std::vector<std::vector<char>> readMatrixFromCSV(const std::string& fileName) {

    std::vector<std::vector<char>> matrix;

    std::ifstream file(fileName);

    std::string line;

    if (!file.is\_open()) {

        std::cerr << "Не удалось открыть файл: " << fileName << std::endl;

        return matrix; // Вернем пустую матрицу в случае ошибки

    }

    while (std::getline(file, line)) {

        std::vector<char> row;

        std::stringstream ss(line);

        std::string cell;

        while (std::getline(ss, cell, ',')) {

            if (!cell.empty()) {

                row.push\_back(cell.at(0)); // Преобразуем строку в символ

            }

        }

        matrix.push\_back(row);

    }

    file.close();

    return matrix;

}

// Для удобства скажем, что матрица квадратная, размера N^2

int maximalRectangle(std::vector<std::vector<char>>& matrix) {

    int rows = matrix.size();

    int cols = matrix[0].size();

    int max\_area = 0;

    for (int y = 0; y < rows; y ++) {       // O(N) итераций

        for (int x = 0; x < cols; x ++) {   // O(N) итераций

            if (matrix[y][x] == '1') {

                int tmp\_max\_add\_y = 1;

                while ((y + tmp\_max\_add\_y) < rows && matrix[y + tmp\_max\_add\_y][x] == '1') { // в среднем О(N / 2) итераций

                    tmp\_max\_add\_y ++;

                }

                for (int tmp\_add\_y = 1; tmp\_add\_y <= tmp\_max\_add\_y; tmp\_add\_y ++) { // в среднем О(N / 2) итераций

                    bool flag = true;

                    int tmp\_area = tmp\_add\_y;

                    for (int j = 1; j < cols - x; j ++) { // в среднем О(N / 2) итераций

                        for (int i = 0; i < tmp\_add\_y; i ++) { // в среднем О(N / 2) итераций

                            if (matrix[y + i][x + j] == '1') { // взятие по индексу - О(1)

                                continue;

                            }

                            else {

                                flag = false;

                                break;

                            }

                        }

                        if (flag) {

                            tmp\_area += tmp\_add\_y;

                        }

                        else {

                            break;

                        }

                    }

                    max\_area = std::max(max\_area, tmp\_area);

                    if (max\_area == rows \* cols)

                        return max\_area;

                }

            }

        }

    }

    return max\_area;

    // Итого асиптотика:

    // O(N) \* O(N) \* O(N / 2) \* O(N / 2) \* O(N / 2) \* О(1) = O(N ^ 5 / 8) - в среднем случае

    // можно апроксимировать до О(N ^ 5), что является худшим случаем

    // Дополнительная память для сложных структур не используется

    // на всех уровнях используются временные int-переменные,

    // использущие О(1) памяти, можно сказать, что память равна асимптотике - О(N ^ 5)

}

int main() {

    // std::vector<std::vector<char>> matrix = {{'1','1','1','1','1','1','1','1'},

    //                                          {'1','1','1','1','1','1','1','0'},

    //                                          {'1','1','1','1','1','1','1','0'},

    //                                          {'1','1','1','1','1','0','0','0'},

    //                                          {'0','1','1','1','1','0','0','0'}};

    // // std::vector<std::vector<char>> matrix = {{'0'}};

    // int max\_area = maximalRectangle(matrix);

    // std::cout << "Result: " << max\_area << std::endl;

    for (int i = 1; i < 100; i ++) {

        std::string fileName = std::to\_string(i) + "x" + std::to\_string(i) + ".csv";

        std::vector<std::vector<char>> matrix = readMatrixFromCSV(fileName);

        auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

        int max\_area = maximalRectangle(matrix);

        auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

        std::chrono::duration<double> duration = end - start;

        //std::cout << (matrix[0][0] == '1') << std::endl;

        // for (const auto& row : matrix) {

        //     for (char val : row) {

        //         std::cout << static\_cast<int>(val) << " "; // Преобразуем char в int для вывода

        //     }

        //     std::cout << std::endl;

        // }

        std::cout << duration.count() << std::endl;

    }

}