ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу

Смирнов Олег

Академическая группа J3110

Принято

Вершинин Владислав

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель: реализовать алгоритм, который находит длину наибольшего возрастающего пути в матрице.

Задачи: найти быстрый алгоритм для поиска максимальной длины, провести неасимптотическую оптимизацию (быстрые структуры данных, исключить лишние затраты по памяти).

1. Теоретическая подготовка

Концепция динамического программирования: не пересчитывать заново, а хранить значения в памяти.

BFS и необходимые для него структуры данных: использую очередь (быстрая вставка в конец и удаление из начала, FIFO) и пару (экономим память).

1. Реализация
2. Определил размер матрицы и создал массив для хранения результатов.
3. Создал массивы di dj для индексов соседей (лишняя память, но код становится читаемым).
4. Подготовил очередь и отсортировал элементы матрицы по возрастанию.
5. Заполнил очередь единичками (индекс минимального числа идёт в начало)
6. Написал BFS с дополнительными условиями
7. Нашёл максимальный результат
8. Экспериментальная часть

Все тесты прошли.

1. Заключение

В ходе работы мною был реализован дп алгоритм для поиска максимального возрастающего пути в матрице. Ожидания от скорости работы алгоритма (O(mn\*log(mn)) совпали с реальностью. В качестве улучшений можно написать сортировку самостоятельно, так как возможно quick sort здесь не самый удачный выбор. Также можно заняться более серьёзной неасимптотической оптимизацией.

ПРИЛОЖЕНИЕ А

Код с подсчётом памяти и времени

class Solution {

public:

    int longestIncreasingPath(vector<vector<int>>& matrix) {

        int n = matrix.size();

        int m = matrix[0].size();

        vector<vector<int>> results(n, vector<int>(m, 1)); //O(m\*n) memory

        vector <int> di = {0, 0, 1, -1};

        vector <int> dj {1, -1, 0, 0};

        queue<pair<pair<int, int>, int>> q;

        vector<pair<int, int>> v;

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < m; j++) {

                v.push\_back({i, j});

            }

        }

        //O(mn\*log(mn))

        sort(v.begin(), v.end(), [&](pair<int, int> a, pair<int, int> b) {

            return matrix[a.first][a.second] < matrix[b.first][b.second];

        });

        //O(m\*n) time and memory

        for (auto [i, j] : v) {

            q.push({{i, j}, 1});

        }

        //BFS с приколами O(mn)

        while (!q.empty()) {

            auto [pos, val] = q.front();

            q.pop();

            //4mn

            for (int i = 0; i < 4; i++) {

                int ii = pos.first + di[i], jj = pos.second + dj[i];

                if (ii < n && ii >= 0 && jj < m && jj >= 0 && matrix[ii][jj] > matrix[pos.first][pos.second] && results[ii][jj] <= val) {

                    results[ii][jj] = val + 1;

                    q.push({{ii, jj}, results[ii][jj]});

                }

            }

        }

        long long mx = 1;

        //O(mn)

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < m; j++) {

                mx = max(mx, 1LL \* results[i][j]); //чтобы не вылетала ошибка из-за разных типов данных

            } cout << '\n';

        }

        return mx;

    }

};

Скриншот, что все тесты прошли

