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

Отчёт по лабораторной работе № 6
«Динамическое программирование»

Выполнил работу
Дышлевский Игорь
Академическая группа №J3111
Принято
Ментор, Владислав Вершинин

Санкт-Петербург
2024

1. Введение

Цель: решить задачу leetcode с помощью Динамического программирования.

Задачи:

- Найти задачу на leetcode
- Реализовать решение
- Провести тесты
- Построить графики скорости работы и сравнить с теоретической оценкой
- Написать отсчет

2. Теоретическая подготовка

Типы данных:

- Int
- Vector

3. Реализация

В моем решении в основе лежит подход Динамического программирования мемоизация (сохранения результатов предыдущих расчетов, чтобы не нужно было пересчитывать). В качество массива с решениями здесь dp, который сохраняет уже посчитанные последовательности и расстояния. Для самого подсчета возрастающих последовательностей я использую модифицированный dfs (тут критерием возможности перехода будет то, что следующий элемент больше предыдущего).

```
int dfs(std::vector<std::vector<int>>& matrix, std::vector<std::vector<int>>& dp
, int row, int col, int prev_val) {
    if (row < 0 || row >= matrix.size() || col < 0 || col >= matrix[0].size() ||
 matrix[row][col] <= prev_val) {
        return 0;
    }
    if (dp[row][col] != -1) {
        return dp[row][col];
    }
    int cur_value = matrix[row][col];
    int left = dfs(matrix, dp, row, col - 1, cur_value);
    int right = dfs(matrix, dp, row, col + 1, cur_value);
    int up = dfs(matrix, dp, row - 1, col, cur_value);
    int down = dfs(matrix, dp, row + 1, col, cur_value);
    dp[row][col] = 1 + std::max({left, right, up, down});
    return dp[row][col];
}

int longestIncreasingPath(std::vector<std::vector<int>>& matrix) {
    int rows = matrix.size();
    int cols = matrix[0].size();
    std::vector<std::vector<int>> dp(rows, std::vector<int>(cols, -1));
    int max_path = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            max_path = std::max(max_path, dfs(matrix, dp, i, j, -1));
        }
    }
    return max_path;
}
```

Рисунок 3.1 Реализация алгоритма

Тесты были реализованы и успешно пройдены. Они вынесены в отдельную функцию с выводом результатов тестирования (OK/Error).

```
void test() {
    std::vector<std::vector<int>> arr = {{9, 9, 4}, {6, 6, 8}, {2, 1, 1}};
    if (longestIncreasingPath(arr) == 4) {
        std::cout << "OK\n";
    } else {
        std::cout << "Test Failed\n";
    }
    arr = {{3, 4, 5} ,{3, 2, 6}, {2, 2, 1}};
    if (longestIncreasingPath(arr) == 4) {
        std::cout << "OK\n";
    } else {
        std::cout << "Test Failed\n";
    }

}
```

Рисунок 3.2 Реализация функции теста

4. Экспериментальная часть

Подсчёт по памяти (только для циклов и сложных структур)

*Подсчеты приведены без учета веса самой структуры - только её содержимого

- вес(vector<vector<int>> arr) = n * size_of(vector<int>) = n * m * size_of(int) = n * m * 4 Байт

- вес(vector<vector<int>> dp) = n * size_of(vector<int>) = n * m * size_of(int) = n * m * 4 Байт

Подсчёт асимптотики (только для циклов и сложных структур).

- O(N*M) - асимптотика формируется прохождением dfs по всему полю (за счет техники мемоизации нет необходимости ходить по уже подсчитанным клеткам), далее происходит перебор точек старта и они просто возвращают предподсчитанные значения или досчитывают оставшуюся часть поля

График зависимости времени от числа элементов. Пример выполнения:

Теоретически заданная сложность задачи составляет O(N*M). Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

Размер входного набора	10	20	100	200
Время выполнения программы, с	4.3E-05	5.7E-05	0.001	0.003
O(N*M), с	9.39E-06	3.756e-05	0.001	0.003

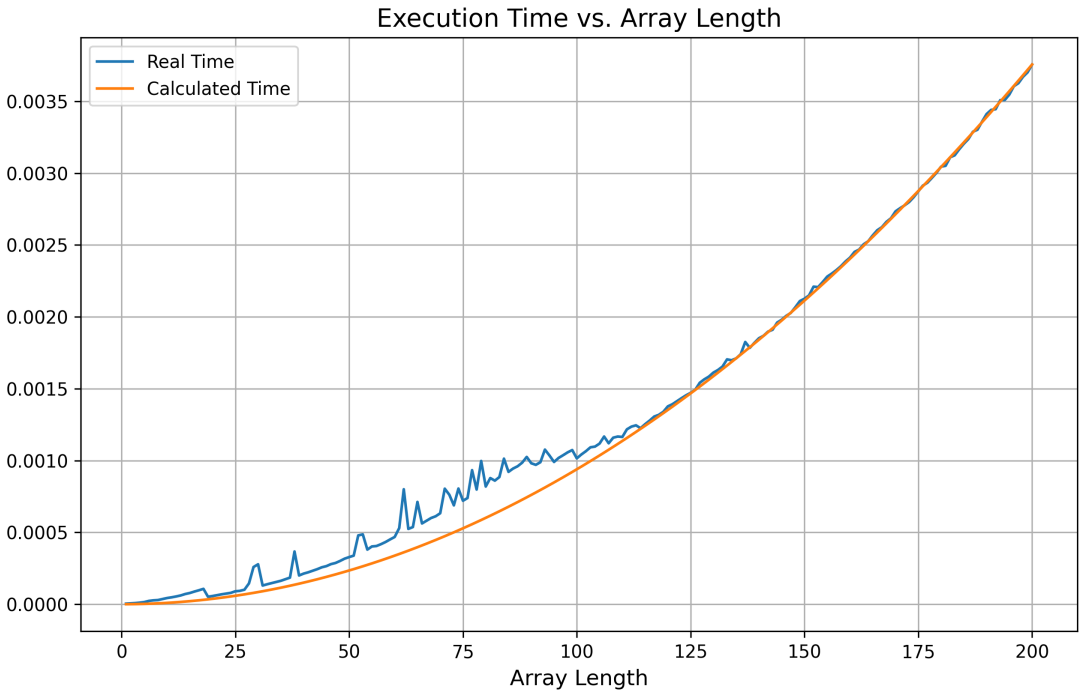

Рисунок 4.1 Теоретическое и реальное время

5. Заключение

В ходе работы были реализован алгоритм с использованием Динамического программирования для решения задачи.

6. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла main6.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>

// using namespace std;


int dfs(std::vector<std::vector<int>>& matrix, std::vector<std::vector<int>>& dp, int row, int col, int prev_val) {
    if (row < 0 || row >= matrix.size() || col < 0 || col >= matrix[0].size() || matrix[row][col] <= prev_val) {
        return 0;
    }
    if (dp[row][col] != -1) {
        return dp[row][col];
    }
    int cur_value = matrix[row][col];
    int left = dfs(matrix, dp, row, col - 1, cur_value);
    int right = dfs(matrix, dp, row, col + 1, cur_value);
    int up = dfs(matrix, dp, row - 1, col, cur_value);
    int down = dfs(matrix, dp, row + 1, col, cur_value);
    dp[row][col] = 1 + std::max({left, right, up, down});
```

```
    return dp[row][col];
}

int longestIncreasingPath(std::vector<std::vector<int>>& matrix) {
    int rows = matrix.size();
    int cols = matrix[0].size();
    std::vector<std::vector<int>> dp(rows, std::vector<int>(cols, -1));
    int max_path = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            max_path = std::max(max_path, dfs(matrix, dp, i, j, -1));
        }
    }
    return max_path;
}


void test() {
    std::vector<std::vector<int>> arr = {{9, 9, 4}, {6, 6, 8}, {2, 1, 1}};
    if (longestIncreasingPath(arr) == 4) {
        std::cout << "OK\n";
    } else {
        std::cout << "Test Failed\n";
    }
    arr = {{3, 4, 5} ,{3, 2, 6}, {2, 2, 1}};
    if (longestIncreasingPath(arr) == 4) {
        std::cout << "OK\n";
    } else {
        std::cout << "Test Failed\n";
    }
```

```
}


int main() {
    test();
    std::vector<std::vector<int>> arr = {{9, 9, 4}, {6, 6, 8}, {2, 1, 1}};
    std::cout << longestIncreasingPath(arr);
    return 0;
}
```