ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу:

Зырянов Виталий

Академическая группа:

J3111

Принято:

Ментор, Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель работы:

Решить задачу на сайте -leetcode, используя принцип динамического программирования.

Задачи:

* Выбрать задачу для решения на leetcode.
* Реализовать решение для задачи с помощью динамического программирования.
* Провести тесты над кодом.
* Определить затрачиваемую память.
* Проанализировать решение посредством графика зависимости скорости работы от входных данных. Сравнить асимптотическую сложность с затрачиваемым временем.

1. Теоретическая подготовка

Используемые типы данных:

* int - целое число, используемый в работе для индексов элементов массивов и в работе самих элементов.
* char – символ, позволяет задавать один символ.
* vector - контейнер, который хранит элементы массива и подмассива.

Используемые функции:

* random\_device -  это компонент стандартной библиотеки C++, который используется для получения случайных чисел.
* chrono::high\_resolution\_clock - функция библиотеки chrono, используемая для получения текущего времени.
* max - функция стандартной библиотеке iostream, позволяет узнать самый большой элемент из двух элементов.
* min - функция стандартной библиотеке iostream, позволяет узнать самый большой элемент из двух элементов.

1. Реализация

В качестве задачи я выбрал определения максимальной площади прямоугольника в матрица, состоящего только из единиц. Решение состоит из принципа динамического программирования. Сначала алгоритм записывает в строчках количество непрерывно подряд идущих единиц. Далее определяем для текущего состояние количество непрерывно подряд идущих единиц и проходим циклом по предыдущим строкам, чтобы определить ширину прямоугольника. На каждой итерации обновляем ответ на задачу. Таким образом работает данное решение. Динамическое программирование помогает решить эту задачу, не затрагиваю сложные оптимизационные алгоритмы.

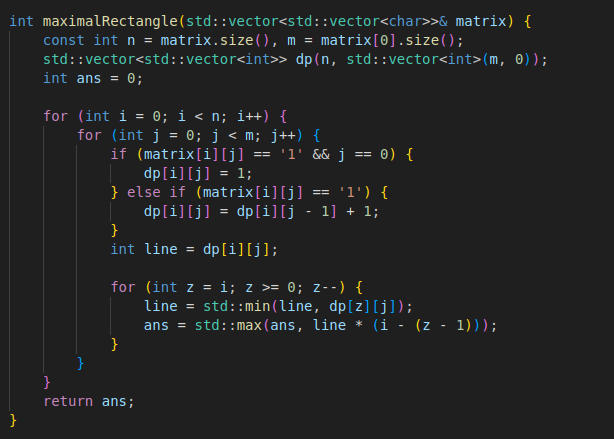


Рисунок. 1 – Алгоритм решения

1. Экспериментальная часть

На данном этапе работы проводилась оценка затрачиваемой памяти, асимптотики и времени программы. Замер времени производился с помощью библиотеки chrono. На основе времени был построен график зависимости размера входного массива и времени выполнения.

Подсчёт памяти:

Вес(vector<vector<int>> dp) = n \* msize\_of(int) = 4nm байт

Подсчёт асимптотики:

O(n\*m) - 2 вложенных цикла (Рисунок 1)

Для тестирования алгоритма была собрана статистика, на основе которой построили графики зависимости входных данных и времени выполнения.

Таблица 1 - Подсчёт сложности для подразрядной сортировки.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество строк матрицы, N | Количество строк матрицы, M | Время выполнения, с: | O(n\*m), c: |
| 10 | 10 | 1.4201e-05 | 9e-05 |
| 20 | 20 | 5.2466e-05 | 0.00036 |
| 50 | 50 | 0.000816582 | 0.00225 |
| 100 | 100 | 0.00432557 | 0.009 |
| 150 | 150 | 0.0176785 | 0.02025 |
| 200 | 200 | 0.0380594 | 0.036 |

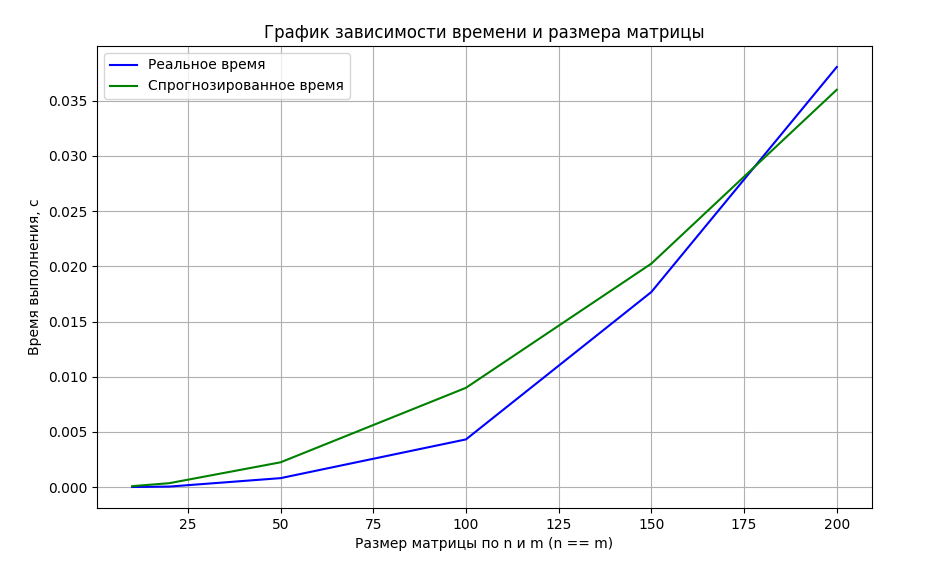


Рисунок. 2 – График работы алгоритма

Анализируя графики на можно увидеть, что сложность алгоритма была определена корректно.

1. Заключение

Входе выполнения лабораторной работы мною был реализован алгоритм, позволяющий определить самый большой по площади прямоугольник в матрице, использую принцип динамического программирования.

1. Приложение

ПРИЛОЖЕНИЕ А

Листинг кода файла task.cpp

#include <iostream>

#include <vector>

#include <cassert>

#include <random>

#include <chrono>

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

const int n = matrix.size(), m = matrix[0].size();

std::vector<std::vector<int>> dp(n, std::vector<int>(m, 0));

int ans = 0;

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

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

if (matrix[i][j] == '1' && j == 0) {

dp[i][j] = 1;

} else if (matrix[i][j] == '1') {

dp[i][j] = dp[i][j - 1] + 1;

}

int line = dp[i][j];

for (int z = i; z >= 0; z--) {

line = std::min(line, dp[z][j]);

ans = std::max(ans, line \* (i - (z - 1)));

}

}

}

return ans;

}

std::vector<std::vector<char>> randMatrix(const int n, const int m) {

std::vector<std::vector<char>> matrix(n, std::vector<char>(m));

std::random\_device rd;

std::mt19937 seed(rd());

std::vector<char> symb = {'0', '1'};

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

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

matrix[i][j] = symb[seed() % 2];

}

}

return matrix;

}

void display(std::vector<std::vector<char>>& matrix, const int& ans) {

for (std::vector<char> i : matrix) {

for (char j : i) {

std::cout << j;

}

std::cout << "\n";

}

}

bool check(const int& ans, const int& true\_ans) {

return ans == true\_ans;

}

void test(std::vector<std::vector<char>>& matrix, const int& true\_ans) {

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

const int ans = maximalRectangle(matrix);

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

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

// std::cout << duration.count() << "\n";

assert(check(ans, true\_ans));

}

int main() {

std::vector<std::vector<char>> matrix1 = {{'1','0','1','0','0'},

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

{'1','1','1','1','1'},

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

test(matrix1, 6);

std::vector<std::vector<char>> matrix2 = {{'1','1','1','0','0'},

{'1','1','1','1','1'},

{'1','1','1','1','1'},

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

test(matrix2, 10);

std::vector<std::vector<char>> matrix3 = {{'1','1','1','0','0','0'},

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

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

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

test(matrix3, 6);

return 0;

}