Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики

Кафедра Теорії оптимальних процесів

Лабораторна робота №3

МНОЖЕННЯ МАТРИЦЬ. АЛГОРИТМ ШТРАССЕНА.

з курсу “Алгоритми обчислювальних процесів ”

Виконав:
студент групи ПМА-11
Ковтун Віталій Олегович

Львів - 2024

Реалізація

У даному досліджені було реалізовано два алгоритми множення матриць: стандартний та алгоритм Штрассена.

- Стандартний метод (функція `multiply()`) показує часову складність O(n^3), для обчислення використовується три вкладених цикли.

- Метод Штрассена (функція `strassen()`) показує часову складність O(n^2.81).

  Реалізується за допомогою семи рекурсивних викликів, що ефективно знижує кількість операцій порівняно з класичним методом. Після отримання проміжних результатів, вони комбінуються для отримання кінцевого результату.

Приклади

Приклад 1.

Дано дві матриці розміру 128 x 128, заповнені випадковими числами. Алгоритм Штрассена стає ефективним вже на матрицях таких розмірів.

```
Enter the size of the matrix:
128
Time taken by Standard algorithm: 8 milliseconds
Time taken by Strassen algorithm: 7 milliseconds
```

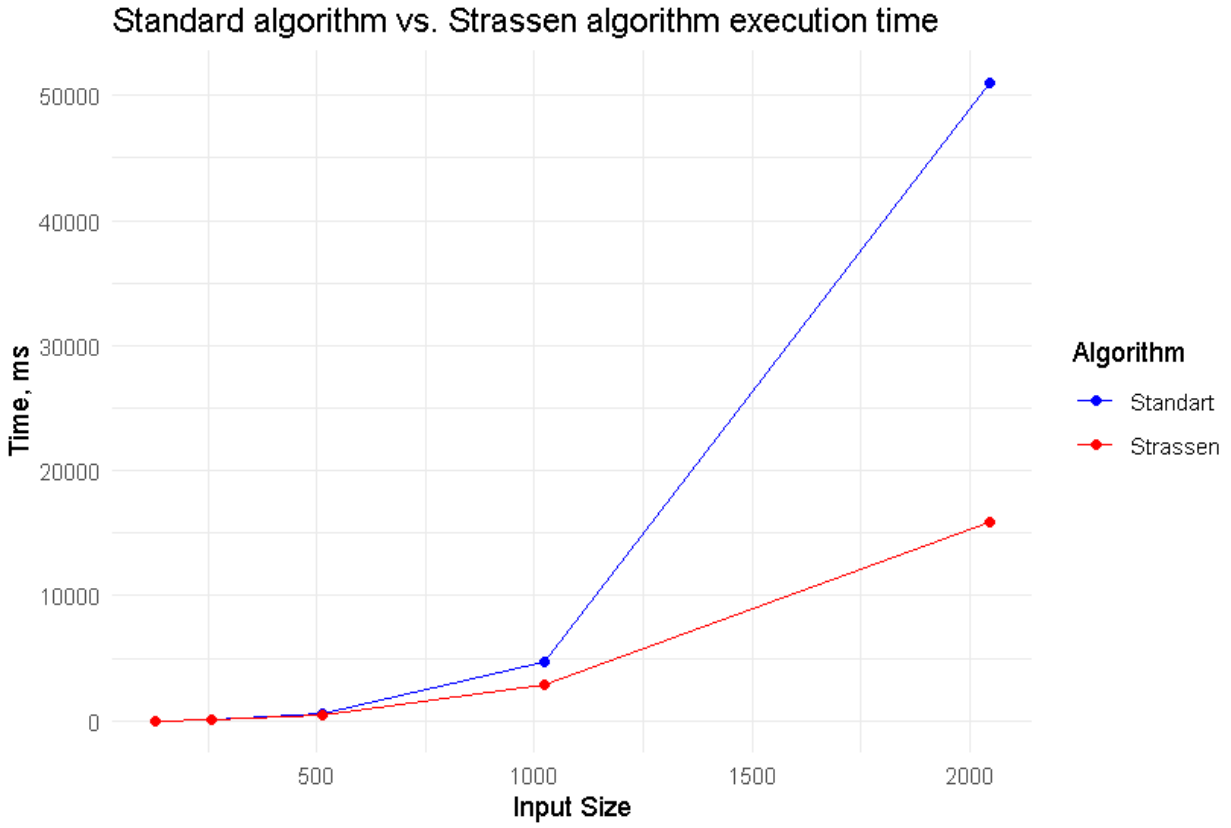
Приклад 2.

Дано дві матриці розміру 1024 x 1024, заповнені випадковими числами.

```
Enter the size of the matrix:
1024
Time taken by Standard algorithm: 4561 milliseconds
Time taken by Strassen algorithm: 2916 milliseconds
```

Загалом було проведено кілька тестів для n == 2^k:

128	8	7
256	61	58
512	555	420
1024	4772	2992
2048	51045	15934



Відносно стандартного алгоритму на матрицях невеликих розмірів (до ~ n == 128) алгоритм Штрассена не є ефективним, зі зростанням розміру матриці ефективність алгоритму відчутно зростає.

А для розмірів, які не є степенем двійки, алгоритм потребує додаткових операцій (додавання рядків/стовпців, заповнених нулями).

Тестування програми

Для тестування алгоритмів було використано бібліотеку cassert(`#include <cassert>`) та створено функцію runTest():

```
void runTest() {
    {
        const int n = 3;
        int** A = createMatrix(n);
        int** B = createMatrix(n);
        int** result;

        randomize(B, n);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                A[i][j] = 0;
            }
        }
        result = multiply(A, B, n);

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                assert(result[i][j] == 0);
            }
        }

        remove(A, n);
        remove(B, n);
        remove(result, n);
    }
    {
        const int n = 3;
        int** A = createMatrix(n);
        int** B = createMatrix(n);
        int** result;

        randomize(A, n);

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                A[i][j] = 0;
            }
        }
        result = multiply(A, B, n);


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

```
                assert(result[i][j] == 0);
            }
        }
        remove(A, n);
        remove(B, n);
        remove(result, n);
    }

    {
        int** A = createMatrix(1);
        int** B = createMatrix(1);
        A[0][0] = 3;
        B[0][0] = 2;

        int** result = strassen(A, B, 1);

        assert(result[0][0] == 6);

        remove(A, 1);
        remove(B, 1);
        remove(result, 1);
    }

    cout << "Tests passed successfully!" << endl;

}
```

У функції написано три тести: множення нульової матриці на заповнену числами, множення матриці, заповнену числами, на нульову матрицю і обчислення матриці розміру 1 x 1.

Висновок

Для обчислення матриць невеликого розміру алгоритм Штрассена не є ефективнішим, ніж стандартний алгоритм множення матриць. Алгоритм Штрассена також потребує великої кількості пам’яті через використання рекурсій. Загалом, з часовою складністю O(n^2.81), метод не є найшвидшим з відомих варіантів множення матриць. Також даний метод не є практичним, оскільки без модифікацій працює тільки на матрицях розміру n x n, де n == 2^k.