Universidad Católica San Pablo

PROGRAMA PROFESIONAL

Ciencias de la Computación

TÍTULO DEL TRABAJO

Multiplicación de Matrices

CURSO

Análisis y diseño de algoritmos

ALUMNOS

- Angel Josue Loayza Huarachi

SEMESTRE: V

AÑO: 2022

“El alumno declara haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo

Contenido

1. Notas:

- El proyecto consta de 6 archivos
    - **Source.cpp:** Contiene el archivo main, donde se crean las matrices A, B y C; y de paso se insertan los datos obtenidos de un .txt. Además, la llamada a las dos funciones de los algoritmos junto a un contador de tiempo. Postada: la información de la matriz A y de matriz B tienen que estar en 2 “documentos de texto” diferentes
    - **GeneralFunctions.hpp:** contiene funciones adicionales y necesarias para el funcionamiento de ambos algoritmos, como *initializeMatriz* (crea la matriz con memoria dinamica), *printMatrix* (imprime en consola la matriz), etc.
    - **Clasicc.hpp:** Contiene el algoritmo de multiplicación de matrices por el método clásico
    - **Strassen.hpp:** Contiene el algoritmo de multiplicación de matrices por el método de Strassen
    - **matrizA.txt:** Contiene todos los datos de la matriz A separados por un espacio
    - **matrizB.txt:** Contiene todos los datos de la matriz B separados por un espacio
- El proyecto se desarrolló en Visual Studio Community
- El proyecto implementado solo acepta multiplicaciones de matrices cuadradas (n x n). Ejemplo: $matrizA(8\ x\ 8)\ X\ matrizB(8\ x\ 8)$
- Exclusivamente el algoritmo Strassen, solo acepta multiplicaciones de matrices de dimensión 2^n. Ejemplo:
    - $matrizA(2\ x\ 2)\ X\ matrizB(2\ x\ 2)$
    - $matrizA(4\ x\ 4)\ X\ matrizB(4\ x\ 4)$
    - $matrizA(8\ x\ 8)\ X\ matrizB(8\ x\ 8)$
    - $matrizA(16\ x\ 16)\ X\ matrizB(16\ x\ 16)$,
    - etc.
- En la línea 15 de Source.cpp se tiene que especificar de que dimensión “n” son las matrices
- El proyecto se subió a GitHub a través de este link:

https://github.com/angel452/ADA

2. Algoritmo Multiplicación Clásica

```cpp
#include <iostream>
#include "GeneralFunctions.hpp"
#pragma once
using namespace std;

void MultiplicacionMatrizClasico(int** A, int** B, int** C, int n)
{
    //Multiplicacion
    for (int row = 0; row < n; row++)
    {
        for (int col = 0; col < n; col++)
        {
            for (int inner = 0; inner < n; inner++)
            {
                C[row][col] += A[row][inner] * B[inner][col];
            }
        }
    }

    cout << "Resultado:" << endl;
    printMatrix(C, n);

}
```


```cpp
#include <iostream>
#include "GeneralFunctions.hpp"
#pragma once
using namespace std;

void MultiplicacionMatrizClasico(int** A, int** B, int** C, int n)
{
    //Multiplicacion
    for (int row = 0; row < n; row++)
    {
        for (int col = 0; col < n; col++)
        {
            for (int inner = 0; inner < n; inner++)
            {
                C[row][col] += A[row][inner] * B[inner][col];
            }
        }
    }

    cout << "Resultado:" << endl;
    printMatrix(C, n);
}
```

3. Algoritmo Multiplicación Strassen

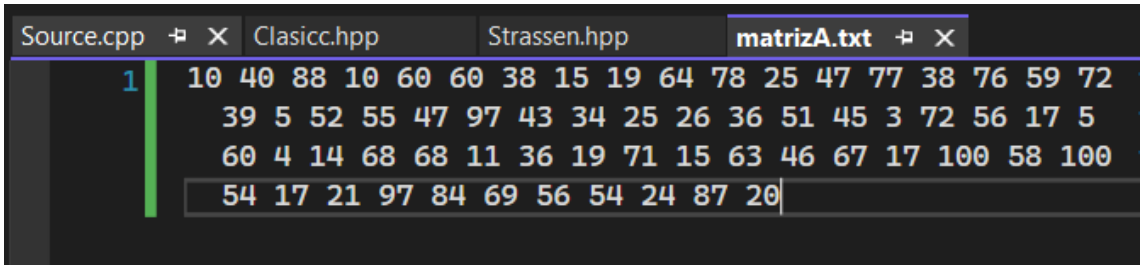
```cpp
#include <iostream>
#include "GeneralFunctions.hpp"
#pragma once

int** strassenMultiply(int** A, int** B, int n)
{
    if (n == 1) {
        int** C = initializeMatrix(1);
        C[0][0] = A[0][0] * B[0][0];
        return C;
    }

    int** C = initializeMatrix(n);
    int k = n / 2;

    int** A11 = initializeMatrix(k);
    int** A12 = initializeMatrix(k);
    int** A21 = initializeMatrix(k);
    int** A22 = initializeMatrix(k);
    int** B11 = initializeMatrix(k);
    int** B12 = initializeMatrix(k);
    int** B21 = initializeMatrix(k);
    int** B22 = initializeMatrix(k);

    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][k + j];
            A21[i][j] = A[k + i][j];
            A22[i][j] = A[k + i][k + j];
            B11[i][j] = B[i][j];
            B12[i][j] = B[i][k + j];
            B21[i][j] = B[k + i][j];
            B22[i][j] = B[k + i][k + j];
        }

    int** P1 = strassenMultiply(A11, subtract(B12, B22, k), k);
    int** P2 = strassenMultiply(add(A11, A12, k), B22, k);
    int** P3 = strassenMultiply(add(A21, A22, k), B11, k);
    int** P4 = strassenMultiply(A22, subtract(B21, B11, k), k);
    int** P5 = strassenMultiply(add(A11, A22, k), add(B11, B22, k), k);
    int** P6 = strassenMultiply(subtract(A12, A22, k), add(B21, B22, k), k);
    int** P7 = strassenMultiply(subtract(A11, A21, k), add(B11, B12, k), k);

    int** C11 = subtract(add(add(P5, P4, k), P6, k), P2, k);
    int** C12 = add(P1, P2, k);
    int** C21 = add(P3, P4, k);
    int** C22 = subtract(subtract(add(P5, P1, k), P3, k), P7, k);

    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            C[i][j] = C11[i][j];
            C[i][j + k] = C12[i][j];
            C[k + i][j] = C21[i][j];
            C[k + i][k + j] = C22[i][j];
        }
    }
    return C;
}
```

```cpp
#include <iostream>
#include "GeneralFunctions.hpp"
#pragma once

int** strassenMultiply(int** A, int** B, int n)
{
    if (n == 1) {
        int** C = initializeMatrix(1);
        C[0][0] = A[0][0] * B[0][0];
        return C;
    }

    int** C = initializeMatrix(n);
    int k = n / 2;

    int** A11 = initializeMatrix(k);
    int** A12 = initializeMatrix(k);
    int** A21 = initializeMatrix(k);
    int** A22 = initializeMatrix(k);
    int** B11 = initializeMatrix(k);
    int** B12 = initializeMatrix(k);
    int** B21 = initializeMatrix(k);
    int** B22 = initializeMatrix(k);

    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][k + j];
            A21[i][j] = A[k + i][j];
            A22[i][j] = A[k + i][k + j];
            B11[i][j] = B[i][j];
            B12[i][j] = B[i][k + j];
            B21[i][j] = B[k + i][j];
            B22[i][j] = B[k + i][k + j];
        }

    int** P1 = strassenMultiply(A11, subtract(B12, B22, k), k);
    int** P2 = strassenMultiply(add(A11, A12, k), B22, k);
    int** P3 = strassenMultiply(add(A21, A22, k), B11, k);
    int** P4 = strassenMultiply(A22, subtract(B21, B11, k), k);
    int** P5 = strassenMultiply(add(A11, A22, k), add(B11, B22, k), k);
    int** P6 = strassenMultiply(subtract(A12, A22, k), add(B21, B22, k), k);
    int** P7 = strassenMultiply(subtract(A11, A21, k), add(B11, B12, k), k);

    int** C11 = subtract(add(add(P5, P4, k), P6, k), P2, k);
    int** C12 = add(P1, P2, k);
    int** C21 = add(P3, P4, k);
    int** C22 = subtract(subtract(add(P5, P1, k), P3, k), P7, k);

    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            C[i][j] = C11[i][j];
            C[i][j + k] = C12[i][j];
            C[k + i][j] = C21[i][j];
            C[k + i][k + j] = C22[i][j];
        }

    for (int i = 0; i < k; i++) {
        delete[] A11[i];
        delete[] A12[i];
        delete[] A21[i];
        delete[] A22[i];
        delete[] B11[i];
```

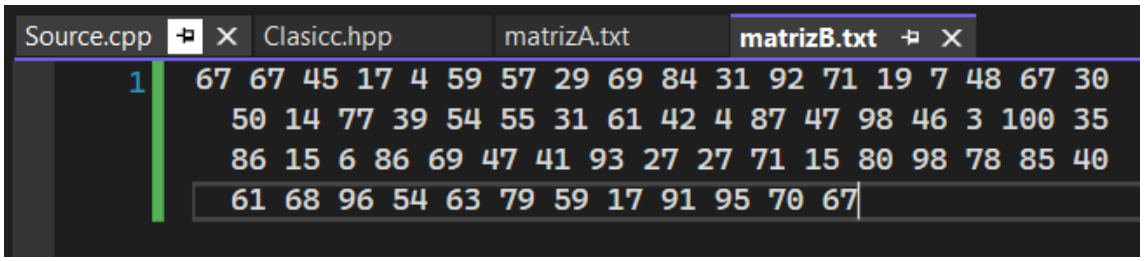
```
            delete[] B12[i];
            delete[] B21[i];
            delete[] B22[i];
            delete[] P1[i];
            delete[] P2[i];
            delete[] P3[i];
            delete[] P4[i];
            delete[] P5[i];
            delete[] P6[i];
            delete[] P7[i];
            delete[] C11[i];
            delete[] C12[i];
            delete[] C21[i];
            delete[] C22[i];
        }

        delete[] A11;
        delete[] A12;
        delete[] A21;
        delete[] A22;
        delete[] B11;
        delete[] B12;
        delete[] B21;
        delete[] B22;
        delete[] P1;
        delete[] P2;
        delete[] P3;
        delete[] P4;
        delete[] P5;
        delete[] P6;
        delete[] P7;
        delete[] C11;
        delete[] C12;
        delete[] C21;
        delete[] C22;

        return C;
    }
```

4. Resultados

- Matriz A de 8 x 8



- Matriz B de 8 x 8



4.1. Algoritmo clásico

- Source.cpp

```
// -------------------- CLASICO ---------------------------------------------


cout << "\t Multiplicacion Clasica " << endl;

//Rellenar matriz C
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        *(*(C + i) + j) = 0;
    }
}

unsigned t0, t1;
t0 = clock();

MultiplicacionMatrizClasico(A,B,C,n);
```

- Resultado

```
Matrix A:
10 40 88 10 60 60 38 15
19 64 78 25 47 77 38 76
59 72 39 5 52 55 47 97
43 34 25 26 36 51 45 3
72 56 17 5 60 4 14 68
68 11 36 19 71 15 63 46
67 17 100 58 100 54 17 21
97 84 69 56 54 24 87 20

Matrix B:
67 67 45 17 4 59 57 29
69 84 31 92 71 19 7 48
67 30 50 14 77 39 54 55
31 61 42 4 87 47 98 46
3 100 35 86 15 6 86 69
47 41 93 27 27 71 15 80
98 78 85 40 61 68 96 54
63 79 59 17 91 95 70 67

         Multiplicacion Clasica
Resultado:
17305 19889 18305 13677 16729 13881 17340 19507
23962 27339 24309 16336 24819 22107 22358 25610
25147 30260 23700 17679 22745 23233 23062 24567
14812 17511 15336 10637 11708 12562 14978 14645
16006 22971 13710 13618 14058 14266 17228 15713
18306 23982 17948 12561 14977 16663 23358 17937
19987 27654 22184 15430 20127 18487 27534 24472
29730 33791 25468 19679 25122 22486 30087 24900

---------------------- Time --------------------------------
Execution Time (s): 0.059
------------------------------------------------------------
```

4.2. Algoritmo Strassen

- Source.cpp

```
// -------------------- STRASSEN ---------------------------------------------


cout << "\t ----- Multiplicacion Strassen ----- " << endl;

unsigned t0, t1;
t0 = clock();

C = strassenMultiply(A, B, n);

cout << "Multipliction result:" << endl;
printMatrix(C, n);


// ---------------------------------------------------------------------------
```

- Resultado

```
Matrix A:
10 40 88 10 60 60 38 15
19 64 78 25 47 77 38 76
59 72 39 5 52 55 47 97
43 34 25 26 36 51 45 3
72 56 17 5 60 4 14 68
68 11 36 19 71 15 63 46
67 17 100 58 100 54 17 21
97 84 69 56 54 24 87 20

Matrix B:
67 67 45 17 4 59 57 29
69 84 31 92 71 19 7 48
67 30 50 14 77 39 54 55
31 61 42 4 87 47 98 46
3 100 35 86 15 6 86 69
47 41 93 27 27 71 15 80
98 78 85 40 61 68 96 54
63 79 59 17 91 95 70 67

         ----- Multiplicacion Strassen -----
Multipliction result:
17305 19889 18305 13677 16729 13881 17340 19507
23962 27339 24309 16336 24819 22107 22358 25610
25147 30260 23700 17679 22745 23233 23062 24567
14812 17511 15336 10637 11708 12562 14978 14645
16006 22971 13710 13618 14058 14266 17228 15713
18306 23982 17948 12561 14977 16663 23358 17937
19987 27654 22184 15430 20127 18487 27534 24472
29730 33791 25468 19679 25122 22486 30087 24900

---------------------- Time --------------------------------
Execution Time (s): 0.032
------------------------------------------------------------
```