INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ANÁLISIS Y DISEÑO DE ALGORITMOS

PRÁCTICA 05

Rodrigo García Mayorga

2012630554

Introducción

La multiplicación de matrices es una operación fundamental en diversas áreas de la computación, tales como el álgebra lineal, la inteligencia artificial, la visión por computadora y la simulación científica. Tradicionalmente, esta operación se realiza utilizando un enfoque directo o "bruto", que consiste en calcular el producto escalar entre filas y columnas, con una complejidad computacional de O(n3)O(n^3)O(n3) para matrices cuadradas de tamaño n×nn \times nn×n.

Sin embargo, la necesidad de algoritmos más eficientes ha impulsado el desarrollo de métodos alternativos. Uno de los más conocidos es el **algoritmo de Strassen**, propuesto por Volker Strassen en 1969, el cual reduce la complejidad del proceso a aproximadamente O(n2.81)O(n^{2.81})O(n2.81). Este método se basa en una técnica de división y conquista que reduce el número de multiplicaciones necesarias, a costa de un incremento en el número de sumas y restas.

Otro enfoque importante es el método **Divide y Vencerás**, que consiste en descomponer las matrices en submatrices más pequeñas, multiplicarlas recursivamente y combinar los resultados. Aunque este enfoque puede tener la misma complejidad teórica que el método tradicional, en algunos casos ofrece ventajas prácticas y es una base conceptual para métodos más avanzados como el de Strassen.

Esta práctica tiene como objetivo implementar y comparar los tres métodos mencionados para la multiplicación de matrices, evaluando su rendimiento y complejidad, y proporcionando una comprensión más profunda de las estrategias algorítmicas involucradas.

Desarrollo

Debido al gran tamaño del código, se omitió la entrega de capturas de pantalla, el código se ha compartido en el enlace de github en la última sección del reporte.

Este programa en C implementa y compara **tres métodos de multiplicación de matrices cuadradas**:

1. **Método Tradicional (Triple For)**
   Multiplica dos matrices A y B usando el algoritmo clásico con tres bucles anidados.
2. **Divide y Vencerás**
   Divide las matrices en submatrices y realiza llamadas recursivas. Aunque conceptualmente divide el problema para reducir complejidad, **el uso intensivo de memoria dinámica (crear y liberar submatrices en cada nivel recursivo)** introduce una sobrecarga que puede hacerlo más lento que el método tradicional en tamaños pequeños o medianos.
3. **Algoritmo de Strassen**
   Un algoritmo más eficiente teóricamente (O(n^2.81) frente a O(n^3)), también basado en dividir matrices, pero con menos multiplicaciones que el método recursivo clásico. Sin embargo, **también incurre en penalización de tiempo por el manejo constante de memoria dinámica**.

Explicación del código:

1. Inclusión de librerías

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
```

Se incluyen bibliotecas estándar para entrada/salida, manejo de memoria dinámica, medición de tiempo y funciones matemáticas.

2. Estructuras básicas

- crear_matriz(int n) y liberar_matriz(...): Reservan y liberan memoria para matrices dinámicas de tamaño n x n.
- llenar_aleatorio(...): Llena matrices con valores aleatorios entre 0 y 9.
- imprimir_matriz(...): Muestra la matriz por consola.

3. Multiplicación tradicional

```
void multiplicar_tradicional(...)
```

Implementa el algoritmo clásico de triple bucle anidado O(n3)O(n^3)O(n3), donde se calcula cada elemento de la matriz resultado como el producto escalar de una fila por una columna.

4. Divide y vencerás

```
void divide_y_venceras(...)
```

Implementa el enfoque de divide y vencerás:

- Divide las matrices en 4 submatrices (cuadrantes).
- Realiza multiplicaciones recursivas de submatrices.
- Combina los resultados para obtener la matriz resultante C.

Este método tiene la misma complejidad que el tradicional, pero sirve como base conceptual para optimizaciones como Strassen.

5. Método de Strassen

```
void strassen(...)
```

Implementa el algoritmo de Strassen:

- También divide las matrices en submatrices.
- Realiza 7 multiplicaciones recursivas específicas (P1 a P7) en lugar de 8.
- Usa combinaciones de sumas/restas para reducir la cantidad de multiplicaciones.

Este algoritmo reduce la complejidad a aproximadamente O(n2.81)O(n^{2.81})O(n2.81), lo cual mejora el rendimiento en matrices grandes.

6. Función principal (main)

- Solicita al usuario el tamaño n de la matriz (debe ser potencia de 2).
- Crea matrices A, B y C.
- Llena A y B con valores aleatorios.
- Mide y muestra el tiempo de ejecución de los tres métodos: tradicional, divide y vencerás, y Strassen.
- Libera la memoria utilizada.

Nota:

Este código **solo funciona correctamente** cuando el tamaño de la matriz es potencia de 2 (por ejemplo, 2, 4, 8, 16...), ya que los métodos recursivos dividen la matriz en mitades iguales.

Detalles adicionales

- Todas las matrices se crean dinámicamente usando malloc y se liberan con free.
- Se mide el tiempo de ejecución de cada método usando clock() y se imprime.
- El usuario debe ingresar un tamaño que sea potencia de 2 (requisito para que los métodos recursivos funcionen correctamente al dividir matrices).
- Las matrices se llenan con números aleatorios entre 0 y 9.

Aunque **Strassen** y **Divide y Vencerás** tienen ventajas teóricas en complejidad, su implementación en este código **puede ser más lenta que la tradicional en la práctica**, especialmente en tamaños pequeños. Esto se debe a:

- La **costosa creación y liberación de muchas submatrices** con malloc/free en cada llamada recursiva.
- El **manejo de punteros dobles (int**)** que añade cierta latencia.
- La **falta de optimización de memoria cache**, que en el método tradicional puede aprovechar mejor la localidad de datos.

En la siguiente pantalla se verá la comparación de tiempos entre métodos:

```
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> gcc matrices.c -o matrices.exe
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> ./matrices.exe
Ingrese el tamano de la matriz (debe ser potencia de 2): 2

Tiempo metodo tradicional: 0.000000 segundos
Tiempo divide y venceras: 0.000000 segundos
Tiempo metodo de Strassen: 0.000000 segundos
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> ./matrices.exe
Ingrese el tamano de la matriz (debe ser potencia de 2): 4

Tiempo metodo tradicional: 0.000000 segundos
Tiempo divide y venceras: 0.000000 segundos
Tiempo metodo de Strassen: 0.000000 segundos
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> ./matrices.exe
Ingrese el tamano de la matriz (debe ser potencia de 2): 8

Tiempo metodo tradicional: 0.000000 segundos
Tiempo divide y venceras: 0.000000 segundos
Tiempo metodo de Strassen: 0.000000 segundos
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> ./matrices.exe
Ingrese el tamano de la matriz (debe ser potencia de 2): 16

Tiempo metodo tradicional: 0.000000 segundos
Tiempo divide y venceras: 0.001000 segundos
Tiempo metodo de Strassen: 0.001000 segundos
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> ./matrices.exe
Ingrese el tamano de la matriz (debe ser potencia de 2): 32

Tiempo metodo tradicional: 0.000000 segundos
Tiempo divide y venceras: 0.016000 segundos
Tiempo metodo de Strassen: 0.012000 segundos
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> ./matrices.exe
Ingrese el tamano de la matriz (debe ser potencia de 2): 64

Tiempo metodo tradicional: 0.002000 segundos
Tiempo divide y venceras: 0.123000 segundos
Tiempo metodo de Strassen: 0.078000 segundos
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> ./matrices.exe
Ingrese el tamano de la matriz (debe ser potencia de 2): 128

Tiempo metodo tradicional: 0.008000 segundos
Tiempo divide y venceras: 1.082000 segundos
Tiempo metodo de Strassen: 0.472000 segundos
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> ./matrices.exe
Ingrese el tamano de la matriz (debe ser potencia de 2): 256

Tiempo metodo tradicional: 0.113000 segundos
Tiempo divide y venceras: 8.003000 segundos
Tiempo metodo de Strassen: 3.299000 segundos
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5> ./matrices.exe
Ingrese el tamano de la matriz (debe ser potencia de 2): 512

Tiempo metodo tradicional: 0.616000 segundos
Tiempo divide y venceras: 72.956000 segundos
Tiempo metodo de Strassen: 26.788000 segundos
PS C:\Users\4PF87LA_RS7\OneDrive\Documentos\ADA\Practica5>
```

Tabla de valores

	Tiempo en segundos		
n	Método tradicional	DyV	Strassen
2	0.00000	0.00000	0.00000
4	0.00000	0.00000	0.00000
6	0.00000	0.00000	0.00000
8	0.00000	0.00000	0.00000
16	0.00000	0.00000	0.00000
32	0.00000	0.016	0.012
64	0.002	0.123	0.078
128	0.008	1.082	0.472
256	0.113	8.003	3.299
512	0.616	72.956	26.788

Enlace del código (github):

https://github.com/LowisN/ADA_Practica5