Juan Diego Naranjo Sergio Alejandro Cabrera Yimar David Tamayo

Método de la Ingeniería

1. Identificación del Problema:

- Necesidades:

  - Venus requiere saber dónde están las naves enemigas
  - Venus no tiene ninguna manera de saber dónde Marte tiene sus naves para así derrotarlos
  - La solución al problema debe tener precisión para que así Venus pueda derrotar a Marte

- Definición del Problema:

Venus necesita un software con el cual se pueda saber dónde van a estar las naves de Marte, su planeta enemigo.

2. Recopilación de Información:

- Definiciones:

  - Matriz:

Conjunto de números o símbolos algebraicos colocados en líneas horizontales y verticales y dispuestos en forma de rectángulo.

  - Aleatorio:

Que depende del azar (casualidad).

  - Multiplicación entre Matrices:

La multiplicación entre matrices se puede efectuar siempre y cuando ambas matrices sean de la misma dimensión, o en su defecto, que el número de filas de la primera, sea igual al número de columnas de la segunda.

3. Búsqueda de Soluciones Creativas:

- Alternativa 1: Algoritmo de Strassen

Sean A, B dos matrices cuadradas sobre un anillo R. Queremos calcular la matriz C como producto

$$\mathbf{C} = \mathbf{A}\mathbf{B} \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in R^{2^n \times 2^n}$$

Si las matrices A, B no son de tipo 2n x 2n habrá que rellenar lo que falta de filas y columnas con ceros.

Partimos A, B y C en matrices de igual tamaño de bloque

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

Con

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

Entonces

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{aligned}$$

Con esta construcción, no hemos reducido el número de multiplicaciones. Todavía tenemos 8 multiplicaciones para calcular la matriz Ci, j , que es el mismo número de multiplicaciones que se necesitan cuando se usa el método estándar de multiplicación de matrices.

Ahora viene la parte importante. Definimos las matrices de nuevo

$$
\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}
$$

que luego se utilizan para expresar Ci, j en términos de Mk. Debido a nuestra definición de la Mk podemos eliminar una multiplicación de matrices y reducir el número de multiplicaciones a 7 (una multiplicación por cada Mk) y expresar Ci, j como

$$
\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned}
$$

Iteramos n-veces el proceso de división hasta que las submatrices degeneran en números (elementos del anillo R).

- Alternativa 2: Algoritmo Paralelo

  Podemos basarnos en el código secuencial,

```
for (i = 0; i < n; i++) {
        for (j = 0; i < n; j++) {
                c[i][j] = 0;
                for (k = 0; k < n; k++) {
                        c[i][j]  += a[i][k] * b[k][j]
                }
        }
}
```

  Ya que los dos bucles externos son independientes en cada iteración.

  Con n procesadores podemos obtener O (n2)

  Con n2 procesadores O (n)

  Estas implementaciones son óptimas en coste, ya que O (n3) = n x O (n2) = n2 x O (n)

Estas cálculos no incluyen el coste de las comunicaciones.

- Alternativa 3: Implementación directa

Con n2 procesadores, cada procesador calcula un elemento de C, por lo que necesita una fila de A y una columna de B.

Si usamos submatrices, cada procesador deberá calcular una submatriz de C.

Análisis de comunicaciones

Cada uno de los n^2 procesadores recibe una fila de A y una columna de B, y devuelve un elemento:

$$t_{com} = n^2(t_{startup} + 2nt_{data}) + n^2(t_{startup} + t_{data}) =$$
$$= n^2(2t_{startup} + (2n+1)t_{data})$$

Mediante un *broadcast* de las dos matrices podemos ahorrar tiempo, por ejemplo en una tenemos:

$$t_{com} = (t_{startup} + n^2 t_{data}) + n^2(t_{startup} + t_{data})$$

Análisis de computación

Cada procesador realiza n multiplicaciones y n sumas, por lo que tenemos:

$$t_{comp} = 2n$$

Usando una estructura de árbol y n3 procesadores podemos obtener un tiempo de computación de O (log n)

4. TRANSICIÓN DE LA FORMULACIÓN DE IDEAS A LOS DISEÑOS PRELIMINARES

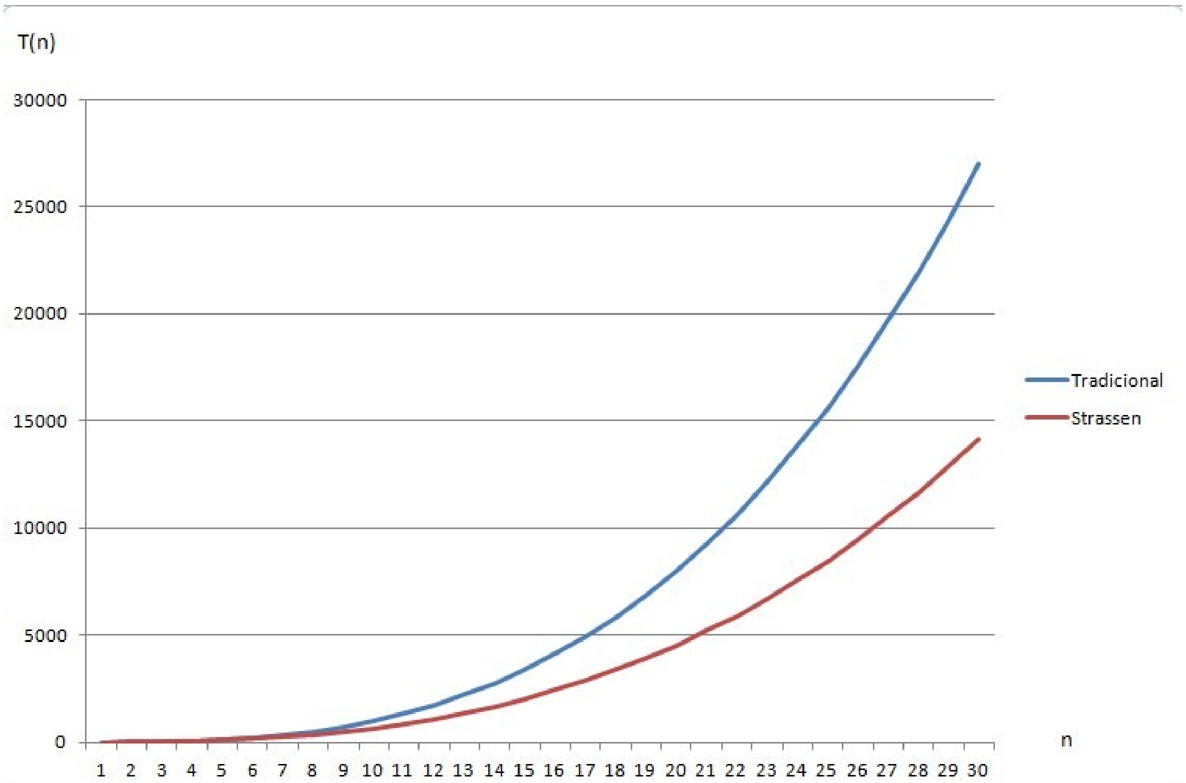
Análisis de complejidad. Resultados:

- Alternativa 1: Algoritmo de Strassen

El orden de este método es de

$O¿$

La reducción en el número de operaciones aritméticas se obtiene a cambio de reducir un tanto la estabilidad numérica.
Diferencia gráfica del método de Strassen y el método tradicional:



- Alternativa 2:

Tal y como se menciona en el planteamiento del método se requiere de uno a 3 procesadores para reducir esa misma cantidad el exponente k de N en $O(N^k)$. Al no ser un método dependiente de la cantidad de operaciones sino de la potencia del hardware hemos decidido que está DESCARTADO.

- Alternativa 3:

Dependiendo del número de procesadores se obtendrá un big O distinto en el caso de n^3el resultado sería de $O(logn)$el cual es muy óptimo pero siendo realistas la cantidad de procesadores es absurda por lo tanto para matrices con un n muy grande el gasto sería excesivo o imposible de soportar.

5.Evaluación y Selección de la Mejor Solución

Criterio A. Precisión de la solución. La alternativa entrega una solución:
[2] Exacta (se prefiere una solución exacta)
[1] Aproximada
Criterio B. Eficiencia. Se prefiere una solución con mejor eficiencia que las otras consideradas. La

eficiencia puede ser:
[4] Constante
[3] Mayor a constante
[2] Logarítmica
[1] Lineal
Criterio C. Completitud. Se prefiere una solución que encuentre todas las soluciones. Cuántas soluciones entrega:
[3] Todas
[2] Más de una si las hay, aunque no todas
[1] Sólo una o ninguna
Criterio D. Facilidad en implementación algorítmica:
[2] Compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno
[1] No compatible completamente con las operaciones aritméticas básicas de un equipo de cómputo moderno

Evaluación

	Criterio A	Criterio B	Criterio C	Criterio D	Total
Algoritmo de Strassen	1	3	2	3	9
Algoritmo en paralelo	2	1	2	1	6
Análisis de computación	1	2	2	1	6

Selección:

Se opta por el método de Strassen por factores como el nivel de los equipos que ejecuten el método así como la rapidez con la cual se realizara el método.

No se tomaron en cuenta los métodos de ***Coppersmith-Winograd, Stothers, Vassilevska Williams*** y ***Francois Le Gall*** ya que requieren un mayor estudio, para el cual el tiempo disponible no fue suficiente.

Especificación de requerimientos funcionales

Nombre	Ingresar los valores de la cantidad de filas y columnas de las dos matrices a multiplicar
Resumen	permita ingresar los valores de la cantidad de filas y columnas de las dos matrices a multiplicar (tablero de batalla)
Entradas	
Valores para las filas y las columnas	
Resultados	

Se recibe los datos y almacenan correctamente

Nombre	Generar aleatoriamente los valores de las posiciones de cada matriz
Resumen	Permite generar aleatoriamente los valores de las posiciones de cada matriz e indica si los números a generar deben ser todos diferentes o pueden haber repetidos.
Entradas	
Identificador que indica si pueden ser diferentes o si puede repetirse.	
Resultados	
Los valores aleatorios indicados son generados correctamente.	

Nombre	Generar una lista aleatoria de matrices
Resumen	Permita generar una lista aleatoria de matrices para posteriormente ser multiplicadas.
Entradas	
Número de matrices que contiene la lista.	
Resultados	
Se crea la lista de matrices éxitosamente.	

Nombre	**Mostrar las posiciones de las naves marcianas**
Resumen	Muestra el resultado de la multiplicación con las posiciones exactas de las tropas de Marte.
Entradas	
Resultados	
Se muestra el resultado de la multiplicación realizada.	

Pseudcódigo de los algoritmos más relevantes

Función para computar el producto de dos matrices cuadradas de tamaño n

```
Matrix matrixMultStrassen(Matrix A, Matrix B)
{
   // Encontrar la dimensión de las matrices (Todas son las mismas)
   int n = A.n;

   // Declarar la nueva matriz que almacena el producto de las matrices A y B
   Matrix C = init_matrix(n);

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

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

            for (int k = 0; k < n; ++k)
            {
                sum = sum + A.content[i][k] * B.content[k][j];
            }

            C.content[i][j] = sum;
        }
    }

    return C;
}
```

Clase matriz

```
struct Matrix
{
    int n;
    int [#][#]content;
};

// Función para inicializar la matriz
Matrix init_matrix(int n)
{
    // Declarar una nueva matriz
    Matrix matrix;

    // Inicializar el tamaño de la matriz
    matrix.n = n;

    // Establecer el tamaño del contenido de la matriz
    matrix.content = new int [#][n];
    for (int i = 0; i < n; ++i)
    {
        matrix.content[i] = new int[n];
    }

    // Inicializar todas las entradas a 0
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            matrix.content[i][j] = 0;
        }
    }

    return matrix;
}
```

Juan Diego Naranjo Sergio Alejandro Cabrera Yimar David Tamayo

Identificar números primos en una matriz

```
function isStrongPseudoprime(n, a)
    d := n - 1; s := 0
    while d % 2 == 0
        d := d / 2
        s := s + 1
    t := powerMod(a, d, n)
    if t == 1 return ProbablyPrime
    while s > 0
        if t == n - 1
            return ProbablyPrime
        t := (t * t) % n
        s := s - 1
    return Composite

function isPrime(n)
    for i from 1 to k
        a := randInt(2, n-1)
        if isStrongPseudoprime(n, a) == Composite
            return Composite
    return ProbablyPrime

function powerMod(b, e, m)
    x := 1
    while e > 0
        if e % 2 == 1
            x := (b * x) % m
        b := (b * b) % m
        e := e // 2 # integer division
    return x
```

Diseño de casos de prueba

Nombre	Clase	Escenario
setupScenary1	Board	*vacío*

Juan Diego Naranjo Sergio Alejandro Cabrera Yimar David Tamayo

setupScenary2	Board	A[][] B[][] $\begin{pmatrix} 5 & 4 & 9 & 2 & 6 \\ 3 & 2 & 6 & 4 & 8 \\ 1 & 6 & 4 & 9 & 5 \\ 2 & 4 & 6 & 9 & 1 \\ 2 & 4 & 9 & 6 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 1 \\ 8 & 6 & 1 & 9 & 4 \\ 7 & 2 & 6 & 5 & 9 \\ 3 & 4 & 8 & 2 & 6 \end{pmatrix}$

Objetivo de la Prueba: Probar que el método de multiplicación de matrices funciona

Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Board	multiply()	SetuScenary1	A[][], B[][]	NullPointerException
Board	multiply()	setuScenary2	A[][], B[][]	$\begin{pmatrix} 133 & 120 & 116 & 159 & 119 \\ 115 & 96 & 119 & 120 & 125 \\ 147 & 106 & 149 & 149 & 138 \\ 140 & 90 & 106 & 145 & 125 \\ 143 & 102 & 91 & 157 & 110 \end{pmatrix}$

Objetivo de la Prueba: Probar que el método de generar matrices random funciona

Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Board	generateMatrixRandom()	SetuScenary1	Int row: 5 Int columns: 5 Int row2: 5 Int columnns2: 5 Boolean	$\begin{pmatrix} 5 & 4 & 9 & 2 & 6 \\ 3 & 2 & 6 & 4 & 8 \\ 1 & 6 & 4 & 9 & 5 \\ 2 & 4 & 6 & 9 & 1 \\ 2 & 4 & 9 & 6 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 1 \\ 8 & 6 & 1 & 9 & 4 \\ 7 & 2 & 6 & 5 & 9 \\ 3 & 4 & 8 & 2 & 6 \end{pmatrix}$

			isEnemy: false	
Board	generateMatrixRandom ()	setuScenary2	Int row: 3 Int columns: 3 Int row2: 3 Int columnns2: 3 Boolean isEnemy: false	$\begin{pmatrix} 8 & 7 & 9 \\ 5 & 6 & 2 \\ 1 & 6 & 8 \end{pmatrix}$

Objetivo de la Prueba: Probar que el método para saber si el número es primo funciona				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Board	isPrimeNumber()	SetuScenary1	Int x: 2	True
Board	isPrimeNumber()	setuScenary2	Int x: 4	False
Board	isPrimeNumber()	setupScenary2	Int x: 1	False

Objetivo de la Prueba: Probar que el método para saber si los números son repetidos funciona				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Board	isRepeated()	SetuScenary1	Int x: 2 Int[][]: null	NullPointerException
Board	isRepeated ()	setuScenary2	Int x: 4 $\begin{pmatrix} 5 & 4 & 9 & 2 & 6 \\ 3 & 2 & 6 & 4 & 8 \\ 1 & 6 & 4 & 9 & 5 \\ 2 & 4 & 6 & 9 & 1 \\ 2 & 4 & 9 & 6 & 1 \end{pmatrix}$	True