MÉTODO DE LA INGENIERÍA

Sergio Martinez (A00045552), Duvan García (A00346605), Juan David Vera Usman (A00293816)

**FASE 1: IDENTIFICACIÓN DEL PROBLEMA**

● Descripción del contexto problemático

Se requiere poder descubrir la posición de las naves de batalla enemigas, el sector de inteligencia interceptará matrices, la matriz interceptada deberá ser comparada con matrices de batallas anteriores almacenadas en su sistema. Para la comparación se realizará multiplicación de matrices procedimiento matemático definido en el **apéndice 1.**

De guerras anteriores se sabe que en las posiciones de batalla (coeficientes dentro de la matriz) indicadas con números primos se hallaban las naves enemigas. La matriz interceptada por inteligencia tendrá coeficientes que cuando sean comparados con matrices de batalla anteriores revelaran posiciones con números primos en la nueva matriz descubriendo así las posiciones de las naves enemigas.

●Identificación del problema

Dada una matriz de batalla A hallar una matriz de batalla C que revele las posiciones enemigas, para esto la matriz de batalla A será comparada con una matriz de batalla B de guerras anteriores. La comparación deberá tardar el menor tiempo posible.

● Requerimientos funcionales y no funcionales

El programa debe estar en la capacidad de:

**RF1** – Identificar cuando dos matrices son multiplicables; compara una matriz Am\*n con una matriz Bq\*p para ver si n = q.

<pre> los coeficientes de las matrices son Ci tal que Ci sea entero >= 0

* Entradas: 2 matrices - salida: valor booleano true para matrices validas

**RF2** – Multiplicar 2 matrices; sigue el proceso de multiplicación del **apéndice 1**

<pre> las matrices son validas

* Entradas: 2 matrices validas - Salida: matriz resultante de la multiplicación

**RF3** – Crear matrices; el usuario puede determinar el tamaño y los valores de una matriz

<pre> solo puede digitar valores >= 0

* Entradas: m y n son números enteros y determinan el número de filas y columnas respectivamente, una colección de números enteros con los valores de los coeficientes.
* Salida: una matriz C

**RF4** – generar aleatoriamente matrices; crea una matriz de tamaños y coeficientes aleatorios

* Entradas: n/a - Salida: matriz Cm\*n

**RF5** – generar colecciones de matrices; una colección de matrices Aij, Bij, Cij …., Nij tal que los tamaños de las matrices puede ser diferentes o iguales

* Entradas: un numero entero <=0 que designara el número total de matrices o el tamaño de la colección
* Salida: una colección de matrices

**RF6** – generar números primos aleatorios; genera un numero primo

* Entradas: n/a - salida: un numero primo

**RNF1** – El tiempo que tome multiplicar matrices deberá ser el más pequeño posible

**RNF2** – Ahorrar espacio en el manejo y almacenamiento de matrices.

**FASE 2: RECOPILACIÓN DE LA INFORMACIÓN NECESARIA**

● Multiplicación de matrices

“En matemática, la multiplicación o producto de matrices es la operación de composición efectuada entre dos matrices, o bien la multiplicación entre una matriz y un escalar según unas determinadas reglas.

Al igual que la multiplicación aritmética, su definición es instrumental, es decir, viene dada por un algoritmo capaz de efectuarla. El algoritmo para la multiplicación matricial es diferente del que resuelve la multiplicación de dos números. La diferencia principal es que la multiplicación de matrices no cumple con la propiedad de conmutatividad.” **Ref. 2**

“Dos matrices A y B son multiplicables si el número de columnas de A coincide con el número de filas de B.”

**Mm x n x Mn x p= M m x p**

**Ref. 1**

● Números primos

“Un número primo es un número entero mayor que cero, que tiene exactamente dos divisores positivos. También podemos definirlo como aquel número entero positivo que no puede expresarse como producto de dos números enteros positivos más pequeños que él, o bien, como producto de dos enteros positivos de más de una forma. Conviene observar que con cualquiera de las dos definiciones el 1 queda excluido del conjunto de los números primos.”**Ref. 3**

“El algoritmo más sencillo que puede utilizarse para saber si un número n es primo es el de la división. Se trata de ir probando para ver si tiene algún divisor propio. Para ello vamos dividiendo el número n entre 2, 3, 4, 5, ... , n-1. Si alguna de las divisiones es exacta (da resto cero) podemos asegurar que el número n es compuesto. Si ninguna de estas divisiones es exacta, el número n es primo. Este método puede hacerse más eficiente observando simplemente, que si un número es compuesto alguno de sus factores (sin contar el 1) debe ser menor o igual que √ n. Por lo tanto, el número de divisiones a realizar es mucho menor. Sólo hay que dividir entre 2, 3, 4, 5, ... , [√ n]. En realidad, bastaría hacer las divisiones entre los números primos menores o iguales que √ n.” **Ref. 3**

● Aleatoriedad de los números primos y aproximación al manejo de las matrices

Para la aleatoriedad en los coeficientes de las matrices se empleará el método “random” de la clase “Math” de java. Para fines académicos manejaremos números mayores que 0 y menores que 100 en la creación de los índices para que cuando sean multiplicados no obtengamos números que tarden mucho tiempo en ser revisados. Además, el conocer los números primos entre 0 y 100 ayudarían a realizar una comprobación más rápida.

Como no es posible realizar la multiplicación de matrices inplace las matrices resultantes deberán ser objetos nuevos que contengan el resultado.

**FASE 3: BÚSQUEDA DE SOLUCIONES CREATIVAS**

Para la búsqueda de alternativas que su puedan utilizar para la solución del problema se realizó una lluvia de idea llevando así a cabo la selección de la mejores. En un fragmento del enunciado dice que la multiplicación de matrices se implementara para darle solución al problema y tiene que ser rápido es decir que los algoritmos no deben tener complejidad muy alta.

**Alternativa 1.  incremento del coeficiente**

Para la multiplicación sabemos que el coeficiente resultante va incrementando en función de la cantidad de multiplicaciones que va realizando es decir: Si Am\*n ^ Bn\*p -> Cm\*p tendrá un Ci = Aij \* Bij + Ai+1,j \* Bi,j+1 + … + Ai+n,j \* Bi,j+n.

Por lo tanto, la estrategia es generar una nueva matriz, la resultante, y a los Ci incrementarlos en función de las multiplicaciones que tienen que realizar.

**Alternativa 2: Separar los procedimientos**

Teniendo claros que hay un paso en el que se multiplica los valores, otro en el que se suma y por último el que asigna los resultados a la nueva matriz. Se realizarán todas las multiplicaciones correspondientes y guardarán en un ArrayList para manipular más fácil los datos, después se realizarán las sumas y se guardarán en una nueva Arraylist por último en una matriz, la resultante, se asignarán los resultados finales.

**Alternativa 3: Operación directa**

Esta estrategia implica asignar en la nueva matriz en los coeficientes correspondientes el resultado directamente realizando todas las operaciones y asignando el resultado a la posición actual, es decir, mientras recorro las posiciones vacías (en 0) de la matriz resultante, opero y agrego el valor correspondiente de una vez.

**Alternativa 4: Separar procedimientos 2.0**

Basando se en la alternativa 3, se realizaría de igual forma, pero empleando arreglos con tamaños definidos

**Alternativa 5:** **Comparar primos**

En esta estrategia la idea es ir revisando si los valores que se van calculando para los coeficientes de una matriz C son primos, si lo son guardar su posición en el tablero de batalla y solo retornar un arreglo con las posiciones ahorrando espacio de memoria vital para matrices de tamaños grandes. Para esta estrategia se usaría la comparación clásica de números primos.

**Alternativa 6:** **Comparar primos 2.0**

Esta estrategia se basa en la anterior, pero en esta para comparar los números primos se usa el teorema pequeño de Fermat que ayuda a verificar números primos más rápidamente mediante una operación matemática.

**FASE 4: TRANSICIÓN DE LA FORMULACIÓN DE IDEAS A LOS DISEÑOS PRELIMINARES**

* **Descarte de ideas no factibles**

Se descarta la alternativa número 6 dado que el teorema pequeño de Fermat no siempre es correcto y pude generar números pseudo primos lo que daría respuestas incorrectas en nuestra solución.

Se descarta la alternativa número 2 dado que los arraylist tienen tamaños reales, en espacio de memoria, mayores que los que presupone el tamaño lógico del método size en arraylist.

* **Diseños preliminares**

Análisis de alternativa 1

A continuación, procedemos a analizar la alternativa 1, se presenta a continuación el seudocódigo del algoritmo para llevar a cabo la multiplicación de esta manera.

|  |  |  |
| --- | --- | --- |
| Método | Temporal | Espacial |
| Multi1(int A [] [], int B [] []) | n/a | m\*n + n\*p |
| Int m = A.lenght | 1 | 1 |
| Int n = A [0].length | 1 | 1 |
| Int p = B [0].length | 1 | 1 |
| int[][] matriz = new int[m][p] | 1 | m\*p |
| For i = 1 to m | m+1 | 1 |
| For j = 1 to p | (m) \*(p+1) | 1 |
| For k = 1 to n | (m) \*(p) \* (n+1) | 1 |
| C[i][j]+=A[i][k] \* B[k][j] | (m) \*(p) \* n | 0 |

T(n,m,p) = 3m + 3P + 2n + 7

Como podemos ver en el seudocódigo el algoritmo tiene tres ciclos, cada uno tiene condiciones de paradas diferentes. Por tanto, el primer ciclo se repite m + 1 veces, el segundo p+1 veces y el tercer ciclo n +1. Por tanto, la complejidad temporal del algoritmo en notación O es T(p\*m\*n) = O (p\*m\*n).

Por otro lado, la complejidad espacial queda representada de la siguiente manera E (m, n, p).

Análisis de alternativa 2

A continuación, procedemos a analizar la alternativa 2, se presenta a continuación el seudocódigo del algoritmo para llevar a cabo la multiplicación de esta manera.

|  |  |  |
| --- | --- | --- |
| m = A.length(), n = B.length(), p = B[0].length() | Complejidad temporal | Complejidad espacial |
| public static int[][] multiplicacionDos(int[][] A, int[][] B){ |  | m\*n+n\*p |
| int [] arreglo = new int[A.length \* B[0].length \* B.length]; | 1 | m\*p\*n |
| int k = 0; | 1 | 1 |
| for(int i = 0; i < A.length; i++) { | m + 1 | 1 |
| for( int j = 0; j < B[0].length; j++) | m\*(p+1) | 1 |
| for(int z = 0; z < A[0].length; z++) | m\*p\*(n+1) | 1 |
| arreglo[k] = A[i][z] \* B[z][j]; | m\*p\*n | 0 |
| k++; | m\*p\*n | 0 |
| } | 0 | 0 |
| int[] suma = new int[B.length \* B.length]; | 1 | 2n |
| int z = 0; k = 0; | 1 | 1 |
| for(int i = 0; i < suma.length; i++) { | 2n + 1 | 1 |
| while(k < arreglo.length && z < B.length) { | 2n^2 + 2n | 0 |
| suma[i] += arreglo[k]; | 2n^2 | 0 |
| k++; z++; | 2(2n^2) | 0 |
| } | 0 | 0 |
| z = 0; | 2n | 0 |
| } | 0 | 0 |
| int[][] C = new int [A.length][B[0].length]; | 1 | 0 |
| k = 0; | 1 | m\*p |
| for(int i = 0; i < C.length; i++) { | m+1 | 0 |
| for( int j = 0; j < C[0].length; j++) | m\*p+1 | 1 |
| C[i][j] = suma[k]; | m\*p | 1 |
| k++; | m\*p | 0 |
| } | 0 | 0 |
| return C; | 1 | 0 |

Análisis alternativa 3

A continuación, se presenta el análisis de la alternativa 3. Se realiza tanto complejidad temporal como espacial

|  |  |  |
| --- | --- | --- |
| Método | Complejidad temporal | Complejidad espacial |
| public static int[][] multiplicacionTres(int[][] A, int[][] B){ | N/a | m\*n + n\*p |
| int[][] C = new int[A.length][B[0].length]; | 1 | m\*p |
| int x = 0, y = 0, z = 0, k = 0; | 1 | 1 |
| for(int i = 0; i < C.length; i++){ | m+1 | 1 |
| for(int j = 0; j < C[0].length; j++){ | m\*(p+1) | 1 |
| while(k < B.length){ | m\*p\*(n+1) | 0 |
| z += A[x][k] \* B[k][y]; | m\*p\*n | 0 |
| k++; | m\*p\*n | 0 |
| y++;     k = 0; | m\*p | 0 |
| C[i][j] = z; | m\*p | 0 |
| z = 0; | m\*p | 0 |
| x++;    y = 0; | m | 0 |
| return C; | 1 | 0 |

T(n,m,p) = 9m +7p + 3n + 8.

La complejidad en notación O se expresa como O(m\*p\*n).

Por otro lado, la complejidad espacial queda representada de la siguiente manera E (m, n, p).

**FASE 5: EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCIÓN**

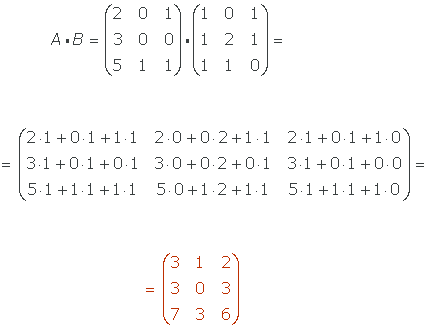
En este caso, se escogieron las alternativas 1,2,3 como mejor solución ya que estas son las que mejor se acomodan a las especificaciones del problema.  Por lo tanto, estas tres soluciones son las que tienen menos complejidad temporal que las demás, así realiza o soluciona el problema en el menor tiempo posible. El tiempo de la alternativa 1 es O(m\*n\*p) la de la 2 es y la de la 3 es O(m\*p\*n) , esto quiere decir que lo realiza en el tiempo más corto posible.

**Apéndice 1.**

Dos matrices A y B son multiplicables si el **número de columnas de A** coincide con el **número de filas de B**.

**Mm x n x Mn x p= M m x p**

El elemento**cij** de la matriz producto se obtiene **multiplicando** cada elemento de la **fila i**de la matriz A por cada elemento de la **columna j**de la matriz B y **sumándolos**.



**Referencias**

**Ref. 1:** <https://www.ditutor.com/matrices/multiplicacion_matrices.html>

**Ref. 2:** <https://es.wikipedia.org/wiki/Multiplicaci%C3%B3n_de_matrices>

**Ref. 3:** <http://mimosa.pntic.mec.es/jgomez53/matema/conocer/primos.htm>