UNIVERSIDAD NACIONAL DE MOQUEGUA

Facultad de Ingeniería y Arquitectura

Escuela Profesional de Ingeniería de Sistemas e Informática

EJERCICIOS DE BIDIMENCIONALES

Estudiante:
Ricardo Jose Arque Chunga

Profesor:
Honorio Apaza Alanoca

8 de diciembre de 2024

Índice

1. Ejercicio 1

El siguiente código Java genera una matriz cuadrada de tamaño $d \times d$ con números aleatorios y calcula la suma de los elementos de la diagonal principal y de la diagonal secundaria.

```java
import java.util.Random;
import java.util.Scanner;

public class SumaDiagonales {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        System.out.print("Ingrese la dimension de la matriz (d): ");
        int dimension = scanner.nextInt();
        int[][] matriz = new int[dimension][dimension];

        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                matriz[i][j] = random.nextInt(100);
            }
        }

        int sumaDiagonalPrincipal = 0;
        int sumaDiagonalSecundaria = 0;

        for (int i = 0; i < dimension; i++) {
            sumaDiagonalPrincipal += matriz[i][i];
            sumaDiagonalSecundaria += matriz[i][dimension - 1 - i];
        }

        System.out.println("Matriz generada:");
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                System.out.print(matriz[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println("Suma de la diagonal principal: " +
    sumaDiagonalPrincipal);
        System.out.println("Suma de la diagonal secundaria: " +
    sumaDiagonalSecundaria);
    }
}
```

Listing 1: Código Java: Suma de Diagonales

Explicación del Código

Este programa tiene varias partes importantes que detallamos a continuación:

- **Importación de bibliotecas**: Se importan las clases `Random` y `Scanner` que son necesarias para generar números aleatorios y para leer la entrada del usuario, respectivamente.
- **Entrada de la dimensión**: Se solicita al usuario que ingrese la dimensión de la matriz cuadrada. Esta dimensión se guarda en la variable `dimension`.
- **Inicialización de la matriz**: Se declara una matriz bidimensional de enteros de tamaño `dimension` por `dimension`.
- **Relleno de la matriz**: Se utiliza un doble bucle `for` para recorrer cada posición de la matriz y rellenarla con números enteros aleatorios entre 0 y 99.
- **Cálculo de las sumas de las diagonales**: Se inicializan dos variables para almacenar las sumas de los elementos de la diagonal principal y secundaria. Luego, se utiliza un bucle `for` para sumar los elementos de ambas diagonales.
- **Impresión de la matriz**: Se imprime la matriz generada en consola utilizando un doble bucle `for`.
- **Salida de las sumas**: Finalmente, se imprimen las sumas de la diagonal principal y la diagonal secundaria.

Complejidad Algorítmica

El análisis de la complejidad del código es el siguiente:

- **Relleno de la matriz**: El tiempo de ejecución para rellenar la matriz con números aleatorios es $O(d^2)$, donde d es la dimensión de la matriz. Esto se debe a que necesitamos recorrer cada elemento de la matriz para asignar un valor aleatorio.
- **Cálculo de las sumas de las diagonales**: El tiempo de ejecución para calcular las sumas de las diagonales es $O(d)$. Esto se debe a que solo necesitamos recorrer una vez cada fila de la matriz para sumar los elementos de las diagonales principal y secundaria.
- **Impresión de la matriz**: El tiempo de ejecución para imprimir la matriz es $O(d^2)$, ya que necesitamos recorrer cada elemento de la matriz para imprimirlo.

Por lo tanto, la complejidad total del algoritmo está dominada por el término $O(d^2)$.
En resumen, el programa tiene una complejidad temporal de:

$$T(d) = O(d^2)$$

donde d es la dimensión de la matriz. Esta complejidad se debe principalmente al tiempo necesario para rellenar e imprimir la matriz, que son operaciones que requieren recorrer todos los elementos de la matriz.

2. Ejercicio 2

El siguiente código Java genera una matriz cuadrada de tamaño $d \times d$ con números aleatorios, la imprime, y luego la rota 90° en el sentido de las agujas del reloj.

```java
import java.util.Scanner;
import java.util.Random;

public class RotacionMatriz {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        System.out.print("Ingrese la dimensi n de la matriz (d): ");
        int dimension = scanner.nextInt();

        int[][] matriz = new int[dimension][dimension];
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                matriz[i][j] = random.nextInt(100);
            }
        }

        System.out.println("Matriz original:");
        imprimirMatriz(matriz);

        int[][] matrizRotada = rotarMatriz(matriz);

        System.out.println("Matriz rotada 90   en el sentido de las
    agujas del reloj:");
        imprimirMatriz(matrizRotada);
    }

    public static int[][] rotarMatriz(int[][] matriz) {
        int dimension = matriz.length;
        int[][] rotada = new int[dimension][dimension];

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

```
34                  rotada[j][dimension - 1 - i] = matriz[i][j];
35              }
36          }
37          return rotada;
38      }
39
40      public static void imprimirMatriz(int[][] matriz) {
41          int dimension = matriz.length;
42          for (int i = 0; i < dimension; i++) {
43              for (int j = 0; j < dimension; j++) {
44                  System.out.print(matriz[i][j] + " ");
45              }
46              System.out.println();
47          }
48      }
49  }
```

Listing 2: Código Java: Rotación de Matriz

Explicación del Código

Este programa tiene varias partes importantes que se detallan a continuación:

- **Importación de bibliotecas**: Se importan las clases `Random` y `Scanner` necesarias para generar números aleatorios y para leer la entrada del usuario, respectivamente.
- **Entrada de la dimensión**: Se solicita al usuario que ingrese la dimensión de la matriz cuadrada. Esta dimensión se guarda en la variable `dimension`.
- **Inicialización y relleno de la matriz**: Se declara una matriz bidimensional de enteros de tamaño `dimension` por `dimension`. Luego, se rellena la matriz con números enteros aleatorios entre 0 y 99 utilizando un doble bucle `for`.
- **Impresión de la matriz original**: Se imprime la matriz generada en consola utilizando la función `imprimirMatriz`.
- **Rotación de la matriz**: Se llama a la función `rotarMatriz` que toma como entrada la matriz original y devuelve una nueva matriz rotada 90° en el sentido de las agujas del reloj.
- **Impresión de la matriz rotada**: Finalmente, se imprime la matriz rotada en consola utilizando la función `imprimirMatriz`.

Complejidad Algorítmica

El análisis de la complejidad del código es el siguiente:

- **Relleno de la matriz**: El tiempo de ejecución para rellenar la matriz con números aleatorios es $O(d^2)$, donde d es la dimensión de la matriz. Esto se debe a que necesitamos recorrer cada elemento de la matriz para asignar un valor aleatorio.
- **Rotación de la matriz**: El tiempo de ejecución para rotar la matriz es $O(d^2)$. La rotación implica recorrer todos los elementos de la matriz para reubicarlos en la matriz rotada.
- **Impresión de la matriz**: El tiempo de ejecución para imprimir la matriz es $O(d^2)$, ya que necesitamos recorrer cada elemento de la matriz para imprimirlo.

Por lo tanto, la complejidad total del algoritmo está dominada por el término $O(d^2)$.
En resumen, el programa tiene una complejidad temporal de:

$$T(d) = O(d^2)$$

donde d es la dimensión de la matriz. Esta complejidad se debe principalmente al tiempo necesario para rellenar, rotar e imprimir la matriz, que son operaciones que requieren recorrer todos los elementos de la matriz.

3. Ejercicio 3

El siguiente código Java genera una matriz cuadrada de tamaño $d \times d$ con números aleatorios, la imprime y luego calcula la suma de los elementos en el perímetro de la matriz.

```java
import java.util.Random;
import java.util.Scanner;

public class PerimetroMatriz {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        System.out.print("Ingrese la dimensi n de la matriz (d): ");
        int dimension = scanner.nextInt();

        int[][] matriz = new int[dimension][dimension];
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                matriz[i][j] = random.nextInt(100);
            }
        }

        System.out.println("Matriz generada:");
        imprimirMatriz(matriz);

        int sumaPerimetro = calcularSumaPerimetro(matriz);
```

```
23
24          System.out.println("Suma de los elementos en el per metro: " +
    sumaPerimetro);
25      }
26
27      public static int calcularSumaPerimetro(int[][] matriz) {
28          int dimension = matriz.length;
29          int suma = 0;
30
31          for (int i = 0; i < dimension; i++) {
32              suma += matriz[0][i]; // Borde superior
33              suma += matriz[dimension - 1][i]; // Borde inferior
34              suma += matriz[i][0]; // Borde izquierdo
35              suma += matriz[i][dimension - 1]; // Borde derecho
36          }
37
38          // Restar las esquinas que se han contado dos veces
39          suma -= (matriz[0][0] + matriz[0][dimension - 1] + matriz[
    dimension - 1][0] + matriz[dimension - 1][dimension - 1]);
40
41          return suma;
42      }
43
44      public static void imprimirMatriz(int[][] matriz) {
45          int dimension = matriz.length;
46          for (int i = 0; i < dimension; i++) {
47              for (int j = 0; j < dimension; j++) {
48                  System.out.print(matriz[i][j] + " ");
49              }
50              System.out.println();
51          }
52      }
53  }
```

Listing 3: Código Java: Cálculo del Perímetro de una Matriz

Explicación del Código

Este programa tiene varias partes importantes que se detallan a continuación:

- **Importación de bibliotecas**: Se importan las clases `Random` y `Scanner` necesarias para generar números aleatorios y para leer la entrada del usuario, respectivamente.
- **Entrada de la dimensión**: Se solicita al usuario que ingrese la dimensión de la matriz cuadrada. Esta dimensión se guarda en la variable `dimension`.
- **Inicialización y relleno de la matriz**: Se declara una matriz bidimensional de enteros de tamaño `dimension` por `dimension`. Luego, se rellena la matriz con números enteros aleatorios entre 0 y 99 utilizando un doble bucle `for`.

- **Impresión de la matriz original**: Se imprime la matriz generada en consola utilizando la función `imprimirMatriz`.
- **Cálculo de la suma del perímetro**: Se llama a la función `calcularSumaPerimetro` que toma como entrada la matriz original y devuelve la suma de los elementos ubicados en el perímetro de la matriz.
- **Impresión de la suma del perímetro**: Finalmente, se imprime la suma de los elementos en el perímetro de la matriz en consola.

Complejidad Algorítmica

El análisis de la complejidad del código es el siguiente:

- **Relleno de la matriz**: El tiempo de ejecución para rellenar la matriz con números aleatorios es $O(d^2)$, donde d es la dimensión de la matriz. Esto se debe a que necesitamos recorrer cada elemento de la matriz para asignar un valor aleatorio.
- **Cálculo de la suma del perímetro**: El tiempo de ejecución para calcular la suma del perímetro es $O(d)$. Esto se debe a que solo necesitamos recorrer una vez cada fila de la matriz para sumar los elementos ubicados en el perímetro.
- **Impresión de la matriz**: El tiempo de ejecución para imprimir la matriz es $O(d^2)$, ya que necesitamos recorrer cada elemento de la matriz para imprimirlo.

Por lo tanto, la complejidad total del algoritmo está dominada por el término $O(d^2)$.
En resumen, el programa tiene una complejidad temporal de:

$$T(d) = O(d^2)$$

donde d es la dimensión de la matriz. Esta complejidad se debe principalmente al tiempo necesario para rellenar e imprimir la matriz, que son operaciones que requieren recorrer todos los elementos de la matriz.

4. Ejercicio 4

El siguiente código Java genera una matriz cuadrada de tamaño $d \times d$ con números aleatorios, la imprime y luego calcula su matriz transpuesta.

```
1 import java.util.Random;
2 import java.util.Scanner;
3
4 public class TranspuestaMatriz {
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
```

```java
        Random random = new Random();

        System.out.print("Ingrese la dimension de la matriz (n): ");
        int dimension = scanner.nextInt();

        int[][] matriz = new int[dimension][dimension];
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                matriz[i][j] = random.nextInt(100);
            }
        }

        System.out.println("Matriz original:");
        imprimirMatriz(matriz);

        int[][] matrizTranspuesta = transponerMatriz(matriz);

        System.out.println("Matriz transpuesta:");
        imprimirMatriz(matrizTranspuesta);
    }

    public static int[][] transponerMatriz(int[][] matriz) {
        int dimension = matriz.length;
        int[][] transpuesta = new int[dimension][dimension];

        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                transpuesta[j][i] = matriz[i][j];
            }
        }
        return transpuesta;
    }

    public static void imprimirMatriz(int[][] matriz) {
        int dimension = matriz.length;
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                System.out.print(matriz[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Listing 4: Código Java: Transposición de Matriz

Explicación del Código

Este programa tiene varias partes importantes que se detallan a continuación:

- **Importación de bibliotecas**: Se importan las clases `Random` y `Scanner` necesarias para generar números aleatorios y para leer la entrada del usuario, respectivamente.
- **Entrada de la dimensión**: Se solicita al usuario que ingrese la dimensión de la matriz cuadrada. Esta dimensión se guarda en la variable `dimension`.
- **Inicialización y relleno de la matriz**: Se declara una matriz bidimensional de enteros de tamaño `dimension` por `dimension`. Luego, se rellena la matriz con números enteros aleatorios entre 0 y 99 utilizando un doble bucle `for`.
- **Impresión de la matriz original**: Se imprime la matriz generada en consola utilizando la función `imprimirMatriz`.
- **Transposición de la matriz**: Se llama a la función `transponerMatriz` que toma como entrada la matriz original y devuelve una nueva matriz que es la transpuesta de la original.
- **Impresión de la matriz transpuesta**: Finalmente, se imprime la matriz transpuesta en consola utilizando la función `imprimirMatriz`.

Complejidad Algorítmica

El análisis de la complejidad del código es el siguiente:

- **Relleno de la matriz**: El tiempo de ejecución para rellenar la matriz con números aleatorios es $O(d^2)$, donde d es la dimensión de la matriz. Esto se debe a que necesitamos recorrer cada elemento de la matriz para asignar un valor aleatorio.
- **Transposición de la matriz**: El tiempo de ejecución para calcular la transpuesta de la matriz es $O(d^2)$. La transposición implica recorrer todos los elementos de la matriz original para ubicarlos en la nueva posición correspondiente en la matriz transpuesta.
- **Impresión de la matriz**: El tiempo de ejecución para imprimir la matriz es $O(d^2)$, ya que necesitamos recorrer cada elemento de la matriz para imprimirlo.

Por lo tanto, la complejidad total del algoritmo está dominada por el término $O(d^2)$.
En resumen, el programa tiene una complejidad temporal de:

$$T(d) = O(d^2)$$

donde d es la dimensión de la matriz. Esta complejidad se debe principalmente al tiempo necesario para rellenar, transponer e imprimir la matriz, que son operaciones que requieren recorrer todos los elementos de la matriz.

5. Ejercicio 5

El siguiente código Java genera una matriz cuadrada de tamaño $n \times n$ con números aleatorios, la imprime y luego verifica si la matriz es simétrica.

```java
import java.util.Random;
import java.util.Scanner;

public class VerificarSimetria {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        System.out.print("Ingrese la dimension de la matriz (n): ");
        int dimension = scanner.nextInt();

        int[][] matriz = new int[dimension][dimension];
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                matriz[i][j] = random.nextInt(100);
            }
        }

        System.out.println("Matriz generada:");
        imprimirMatriz(matriz);

        if (esSimetrica(matriz)) {
            System.out.println("La matriz es simetrica.");
        } else {
            System.out.println("La matriz no es simetrica.");
        }
    }

    public static boolean esSimetrica(int[][] matriz) {
        int dimension = matriz.length;
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                if (matriz[i][j] != matriz[j][i]) {
                    return false;
                }
            }
        }
        return true;
    }

    public static void imprimirMatriz(int[][] matriz) {
        int dimension = matriz.length;
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                System.out.print(matriz[i][j] + " ");
```

```
46                 }
47                 System.out.println();
48             }
49         }
50 }
```

Listing 5: Código Java: Verificación de Simetría de una Matriz

Explicación del Código

Este programa tiene varias partes importantes que se detallan a continuación:

- **Importación de bibliotecas**: Se importan las clases `Random` y `Scanner` necesarias para generar números aleatorios y para leer la entrada del usuario, respectivamente.
- **Entrada de la dimensión**: Se solicita al usuario que ingrese la dimensión de la matriz cuadrada. Esta dimensión se guarda en la variable `dimension`.
- **Inicialización y relleno de la matriz**: Se declara una matriz bidimensional de enteros de tamaño `dimension` por `dimension`. Luego, se rellena la matriz con números enteros aleatorios entre 0 y 99 utilizando un doble bucle `for`.
- **Impresión de la matriz original**: Se imprime la matriz generada en consola utilizando la función `imprimirMatriz`.
- **Verificación de simetría**: Se llama a la función `esSimetrica` que toma como entrada la matriz original y devuelve un valor booleano indicando si la matriz es simétrica o no. Una matriz es simétrica si es igual a su transpuesta.
- **Impresión del resultado**: Finalmente, se imprime en consola si la matriz es simétrica o no.

Complejidad Algorítmica

El análisis de la complejidad del código es el siguiente:

- **Relleno de la matriz**: El tiempo de ejecución para rellenar la matriz con números aleatorios es $O(n^2)$, donde n es la dimensión de la matriz. Esto se debe a que necesitamos recorrer cada elemento de la matriz para asignar un valor aleatorio.
- **Verificación de simetría**: El tiempo de ejecución para verificar si la matriz es simétrica es $O(n^2)$. La verificación implica comparar cada elemento `matriz[i][j]` con su correspondiente `matriz[j][i]`.
- **Impresión de la matriz**: El tiempo de ejecución para imprimir la matriz es $O(n^2)$, ya que necesitamos recorrer cada elemento de la matriz para imprimirlo.

Por lo tanto, la complejidad total del algoritmo está dominada por el término $O(n^2)$.
En resumen, el programa tiene una complejidad temporal de:

$$T(n) = O(n^2)$$

donde n es la dimensión de la matriz. Esta complejidad se debe principalmente al tiempo necesario para rellenar, verificar la simetría e imprimir la matriz, que son operaciones que requieren recorrer todos los elementos de la matriz.

6. Ejercicio 6

El siguiente código Java genera una matriz de tamaño *filas* x *columnas* con números aleatorios, la imprime, y luego realiza un recorrido en espiral de la matriz.

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

public class RecorridoEspiral {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        System.out.print("Ingrese la cantidad de filas de la matriz: ");
        int filas = scanner.nextInt();

        System.out.print("Ingrese la cantidad de columnas de la matriz:
    ");
        int columnas = scanner.nextInt();

        int[][] matriz = new int[filas][columnas];
        for (int i = 0; i < filas; i++) {
            for (int j = 0; j < columnas; j++) {
                matriz[i][j] = random.nextInt(100);
            }
        }

        System.out.println("Matriz generada:");
        imprimirMatriz(matriz);

        List<Integer> recorrido = recorrerEspiral(matriz);

        System.out.println("Recorrido en espiral: " + recorrido);
    }

    public static List<Integer> recorrerEspiral(int[][] matriz) {
        List<Integer> resultado = new ArrayList<>();
```

```
34          if (matriz == null || matriz.length == 0) return resultado;
35
36          int filaInicial = 0;
37          int filaFinal = matriz.length - 1;
38          int columnaInicial = 0;
39          int columnaFinal = matriz[0].length - 1;
40
41          while (filaInicial <= filaFinal && columnaInicial <=
    columnaFinal) {
42              for (int i = columnaInicial; i <= columnaFinal; i++) {
43                  resultado.add(matriz[filaInicial][i]);
44              }
45              filaInicial++;
46
47              for (int i = filaInicial; i <= filaFinal; i++) {
48                  resultado.add(matriz[i][columnaFinal]);
49              }
50              columnaFinal--;
51
52              if (filaInicial <= filaFinal) {
53                  for (int i = columnaFinal; i >= columnaInicial; i--) {
54                      resultado.add(matriz[filaFinal][i]);
55                  }
56              }
57              filaFinal--;
58
59              if (columnaInicial <= columnaFinal) {
60                  for (int i = filaFinal; i >= filaInicial; i--) {
61                      resultado.add(matriz[i][columnaInicial]);
62                  }
63              }
64              columnaInicial++;
65          }
66          return resultado;
67      }
68
69      public static void imprimirMatriz(int[][] matriz) {
70          int filas = matriz.length;
71          int columnas = matriz[0].length;
72          for (int i = 0; i < filas; i++) {
73              for (int j = 0; j < columnas; j++) {
74                  System.out.print(matriz[i][j] + " ");
75              }
76              System.out.println();
77          }
78      }
79  }
```

Listing 6: Código Java: Recorrido en Espiral de una Matriz

Explicación del Código

Este programa tiene varias partes importantes que se detallan a continuación:

- **Importación de bibliotecas**: Se importan las clases `Random`, `Scanner`, `ArrayList` y `List` necesarias para generar números aleatorios, leer la entrada del usuario y manejar la lista de resultado, respectivamente.
- **Entrada de las dimensiones**: Se solicita al usuario que ingrese la cantidad de filas y columnas de la matriz. Estas dimensiones se guardan en las variables `filas` y `columnas`.
- **Inicialización y relleno de la matriz**: Se declara una matriz bidimensional de enteros de tamaño `filas` por `columnas`. Luego, se rellena la matriz con números enteros aleatorios entre 0 y 99 utilizando un doble bucle `for`.
- **Impresión de la matriz original**: Se imprime la matriz generada en consola utilizando la función `imprimirMatriz`.
- **Recorrido en espiral de la matriz**: Se llama a la función `recorrerEspiral` que toma como entrada la matriz original y devuelve una lista de enteros con los elementos de la matriz en orden de recorrido en espiral.
- **Impresión del recorrido en espiral**: Finalmente, se imprime en consola el recorrido en espiral de la matriz.

Complejidad Algorítmica

El análisis de la complejidad del código es el siguiente:

- **Relleno de la matriz**: El tiempo de ejecución para rellenar la matriz con números aleatorios es $O(f \cdot c)$, donde f es el número de filas y c es el número de columnas. Esto se debe a que necesitamos recorrer cada elemento de la matriz para asignar un valor aleatorio.
- **Recorrido en espiral**: El tiempo de ejecución para realizar el recorrido en espiral de la matriz es $O(f \cdot c)$. Esto se debe a que necesitamos recorrer cada elemento de la matriz una vez.
- **Impresión de la matriz**: El tiempo de ejecución para imprimir la matriz es $O(f \cdot c)$, ya que necesitamos recorrer cada elemento de la matriz para imprimirlo.

Por lo tanto, la complejidad total del algoritmo está dominada por el término $O(f \cdot c)$.
En resumen, el programa tiene una complejidad temporal de:

$$T(f, c) = O(f \cdot c)$$

donde f es el número de filas y c es el número de columnas de la matriz. Esta complejidad se debe principalmente al tiempo necesario para rellenar, recorrer en espiral e imprimir la matriz, que son operaciones que requieren recorrer todos los elementos de la matriz.

7. Conclusiones

En este documento, hemos analizado la complejidad algorítmica de seis programas diferentes en Java, cada uno con distintas funcionalidades relacionadas con matrices. A continuación, se presenta un cuadro comparativo que resume la complejidad temporal de cada uno de los ejercicios:

Ejercicio	**Funcionalidad**	**Complejidad T**
Suma de Diagonales	Calcula la suma de las diagonales principal y secundaria	$O(d^2)$
Rotación de Matriz	Rota la matriz 90° en el sentido de las agujas del reloj	$O(d^2)$
Cálculo del Perímetro	Calcula la suma de los elementos en el perímetro	$O(d^2)$
Transposición de Matriz	Calcula la transpuesta de una matriz	$O(d^2)$
Verificación de Simetría	Verifica si la matriz es simétrica	$O(d^2)$
Recorrido en Espiral	Realiza un recorrido en espiral de la matriz	$O(f \cdot c$

Cuadro 1: Cuadro Comparativo de la Complejidad Algorítmica de los Ejercicios

Conclusiones

En esta sección, compararemos y discutiremos la complejidad algorítmica de cada uno de los ejercicios para entender mejor sus características y cómo se comportan en diferentes escenarios.

Suma de Diagonales

El ejercicio de suma de diagonales involucra recorrer una matriz cuadrada $d \times d$ para sumar los elementos de la diagonal principal y la diagonal secundaria. La complejidad temporal de este ejercicio es $O(d^2)$ debido a que, en el peor de los casos, se requiere recorrer todos los elementos de la matriz para sumar las diagonales.

Rotación de Matriz

En el ejercicio de rotación de matriz, se rota una matriz cuadrada 90° en el sentido de las agujas del reloj. La complejidad temporal de este ejercicio también es $O(d^2)$, ya que es necesario acceder y reubicar cada uno de los elementos de la matriz en su nueva posición.

Cálculo del Perímetro

El cálculo de la suma de los elementos en el perímetro de una matriz cuadrada involucra sumar los elementos de las filas y columnas externas. A pesar de que solo se suman los elementos de los bordes, la complejidad sigue siendo $O(d^2)$ debido al acceso a cada uno de los elementos del perímetro, sumado a la consideración de los elementos de las esquinas.

Transposición de Matriz

La transposición de una matriz implica intercambiar filas por columnas. Este ejercicio también tiene una complejidad temporal de $O(d^2)$, ya que es necesario recorrer cada elemento de la matriz para ubicarlo en su nueva posición en la matriz transpuesta.

Verificación de Simetría

Para verificar si una matriz es simétrica, se compara cada elemento $matriz[i][j]$ con $matriz[j][i]$. La complejidad temporal de este ejercicio es $O(d^2)$, dado que cada par de elementos debe ser comparado, recorriendo así todos los elementos de la matriz.

Recorrido en Espiral

El recorrido en espiral de una matriz $f \times c$ involucra acceder a cada elemento de la matriz en un orden específico. La complejidad temporal de este ejercicio es $O(f \cdot c)$, ya que se necesita recorrer todos los elementos una sola vez.

Comparación y Observaciones Finales

El análisis realizado muestra que la mayoría de los ejercicios tienen una complejidad temporal de $O(d^2)$, lo cual es común para operaciones que implican recorrer y manipular todos los elementos de una matriz cuadrada. Sin embargo, el ejercicio de recorrido en espiral destaca por manejar matrices rectangulares y su complejidad está en función del producto de las filas y columnas $O(f \cdot c)$.
Este análisis de complejidad es crucial para entender el rendimiento de los algoritmos en distintos contextos y escalas. Es fundamental considerar la estructura y el tamaño de la matriz al implementar y seleccionar algoritmos, ya que estos factores influirán significativamente en la eficiencia y el tiempo de ejecución del programa.