**Realizado por:**

Andres Felipe Diaz Gonzalez

Juan Camilo Restrepo Velez

Wilder Valencia Ocampo

**4. Análisis de las operaciones del ADT Matriz**

4.1. **La operación suma de matrices**

1. Accesos al arreglo

*public Matriz sumar(IMatriz b) throws Exception {*

*if(Filas != b.getFilas() || Columnas != b.getColumnas()) throw new Exception("Las matrices deben ser del mismo orden");*

*double[][] c = new double[Filas][Columnas],aux=b.getMatriz();*

*for(int i=0; i < Filas; i++)*

*for(int j=0; j < Columnas; j++)*

*c[i][j] = matriz[i][j]+aux[i][j];*

*return new Matriz(c);*

*}*

Considerar

*N = Filas M = Columnas*

|  |  |  |
| --- | --- | --- |
| *Constante* | *Frecuencia* | *Valor Frecuencia* |
| *C1* | *F1* | *N\*M* |
| *C2* | *F2* | *N\*M* |
| *C2* | *F3* | *N\*M* |

*T(N,M)<=C1\*F1+ C2\*F2+ C3\*F3*

*T(N,M)<=C1\* NM + C2\* NM+ C3\* NM*

*T(N,M)<=aNM*

Modelo de costo: 3NM

Función tilde: 3kNM

Orden de crecimiento: Cuadrático (NM)

1. sumas/restas

public Matriz sumar(IMatriz b) throws Exception {

if(Filas != b.getFilas() || Columnas != b.getColumnas()) throw new Exception("Las matrices deben ser del mismo orden");

double[][] c = new double[Filas][Columnas],aux=b.getMatriz();

for(int i=0; i < Filas; i++)

for(int j=0; j < Columnas; j++)

c[i][j] = matriz[i][j]+aux[i][j];

return new Matriz(c);

}

Considerar

N = Filas M = Columnas

|  |  |  |
| --- | --- | --- |
| *Constante* | *Frecuencia* | *Valor Frecuencia* |
| *C1* | *F1* | *N* |
| *C2* | *F2* | *N\*M* |
| *C3* | *F3* | *N\*M* |

T(N)<=C1\*F1+ C2\*F2+ C3\*F3

T(N)<=C1\*N+ C2\* NM+ C3\* NM

T(N)<=aNM+bN

Modelo de costo: 2NM

Función tilde: 2kNM

Orden de crecimiento: Cuadrático (NM)

1. multiplicaciones/divisiones

public Matriz sumar(IMatriz b) throws Exception {

if(Filas != b.getFilas() || Columnas != b.getColumnas()) throw new Exception("Las matrices deben ser del mismo orden");

double[][] c = new double[Filas][Columnas],aux=b.getMatriz();

for(int i=0; i < Filas; i++)

for(int j=0; j < Columnas; j++)

c[i][j] = matriz[i][j]+aux[i][j];

return new Matriz(c);

}

Considerar

N = Filas M = Columnas

Modelo de costo: 0

Función tilde:0

Orden de crecimiento: Constante

4.2. **La operación producto de matrices**

1. Accesos al arreglo

public Matriz multiplicar(IMatriz b) throws Exception {

if(Columnas != b.getFilas()) throw new Exception("Las filas de la primera deben ser iguales a las columnas de la segunda");

double[][] c = new double[Filas][b.getColumnas()], aux = b.getMatriz();

double item = 0;

for(int i=0; i < Filas; i++) {

for(int j=0; j < b.getColumnas(); j++) {

for(int k=0; k < b.getFilas(); k++)

item += matriz[i][k] \* aux[k][j];

c[i][j] = item;

item = 0;

}

}

return new Matriz(c);

}

Considerar

N = Filas M = getColumnas() P = getFilas()

|  |  |  |
| --- | --- | --- |
| *Constante* | *Frecuencia* | *Valor Frecuencia* |
| *C1* | *F1* | *N\*M\*P* |
| *C2* | *F2* | *N\*M\*P* |
| *C3* | *F3* | *N\*M* |

T(N)<=C1\*F1+ C2\*F2+ C3\*F3

T(N)<=C1\*NMP+ C2\*NMP+ C3\* NM

T(N)<=aNMP+bNM

Modelo de costo: 2NMP

Función tilde: 2kNMP

Orden de crecimiento: Cubico (NMP)

1. Sumas y Restas

public Matriz multiplicar(IMatriz b) throws Exception {

if(Columnas != b.getFilas()) throw new Exception("Las filas de la primera deben ser iguales a las columnas de la segunda");

double[][] c = new double[Filas][b.getColumnas()], aux = b.getMatriz();

double item = 0;

for(int i=0; i < Filas; i++) {

for(int j=0; j < b.getColumnas(); j++) {

for(int k=0; k < b.getFilas(); k++)

item += matriz[i][k] \* aux[k][j];

c[i][j] = item;

item = 0;

}

}

return new Matriz(c);

}

Considerar

N = Filas M = getColumnas() P = getFilas()

|  |  |  |
| --- | --- | --- |
| *Constante* | *Frecuencia* | *Valor Frecuencia* |
| *C1* | *F1* | *N* |
| *C2* | *F2* | *N\*M* |
| *C3* | *F3* | *N\*M\*P* |
| *C4* | *F4* | *N\*M\*P* |

T(N)<=C1\*F1+ C2\*F2+ C3\*F3+ C4\*F4

T(N)<=C1\*N+ C2\* NM+ C3\* NMP+ C4\* NMP

T(N)<=aNMP+bNM+cN

Modelo de costo: 2NMP

Función tilde: 2kNMP

Orden de crecimiento: Cubico(NMP)

1. multiplicaciones/divisiones

public Matriz multiplicar(IMatriz b) throws Exception {

if(Columnas != b.getFilas()) throw new Exception("Las filas de la primera deben ser iguales a las columnas de la segunda");

double[][] c = new double[Filas][b.getColumnas()], aux = b.getMatriz();

double item = 0;

for(int i=0; i < Filas; i++) {

for(int j=0; j < b.getColumnas(); j++) {

for(int k=0; k < b.getFilas(); k++)

item += matriz[i][k] \* aux[k][j];

c[i][j] = item;

item = 0;

}

}

return new Matriz(c);

}

Considerar

N = Filas M = getColumnas() P = getFilas()

|  |  |  |
| --- | --- | --- |
| *Constante* | *Frecuencia* | *Valor Frecuencia* |
| *C1* | *F1* | *N\*M\*P* |

T(N)<=C1\*F1

T(N)<=C1\*NMP

T(N)<=aNMP

Modelo de costo: NMP

Función tilde: kNMP

Orden de crecimiento: Cuadrático (NMP)

4.3. **La operación eliminación gaussiana en una matriz**

1. Accesos al arreglo

public double[] eliminacionGauss(double[] b) throws Exception {

if(Filas != Columnas || Filas != b.length) throw new Exception("La matriz debe ser NxN y el tamaño del vector debe ser N");

int k, i , j;

for(i=0; i < Filas; i++)

if(matriz[i][i] == 0) throw new Exception("La diagonal principal no puede contener ceros");

double s = 0;

double[] x = new double[Filas];

double[][] temp = matriz;

for(i = 0; i < Filas - 1; i++) {

for(j = i + 1; j < Filas; j++) {

b[j] = temp[i][i]\*b[j] - temp[j][i]\*b[i];

for(k = Filas - 1; k >= 0; k--)

temp[j][k] = temp[i][i]\*temp[j][k] - temp[j][i]\*temp[i][k];

}

}

for(i = Filas - 1; i >= 0; i--, s = 0) {

for(j = i + 1; j < Filas; j++)

s += temp[i][j]\*x[j];

x[i] = (b[i] - s) / temp[i][i];

}

return x;

}

Considerar

N = Filas

|  |  |  |
| --- | --- | --- |
| *Constante* | *Frecuencia* | *Valor Frecuencia* |
| *C1* | *F1* | *(N-1)\*((N+1)/2)\*N* |
| *C2* | *F2* | *(N-1)\*((N+1)/2)\*N* |
| *C3* | *F3* | *(N-1)\*((N+1)/2)\*N* |
| *C4* | *F4* | *(N-1)\*((N+1)/2)\*N* |

T(N)<=C1\*F1+ C2\*F2+ C3\*F3+ C4\*F4

T(N)<=C1\*() +C2\*()+ C3\*()+ C4\*()

T(N)<=

Modelo de costo: 4

Función tilde: 4k

Orden de crecimiento: Cubico ()

1. sumas/restas

public double[] eliminacionGauss(double[] b) throws Exception {

if(Filas != Columnas || Filas != b.length) throw new Exception("La matriz debe ser NxN y el tamaño del vector debe ser N");

int k, i , j;

for(i=0; i < Filas; i++)

if(matriz[i][i] == 0) throw new Exception("La diagonal principal no puede contener ceros");

double s = 0;

double[] x = new double[Filas];

double[][] temp = matriz;

for(i = 0; i < Filas - 1; i++) {

for(j = i + 1; j < Filas; j++) {

b[j] = temp[i][i]\*b[j] - temp[j][i]\*b[i];

for(k = Filas - 1; k >= 0; k--)

temp[j][k] = temp[i][i]\*temp[j][k] - temp[j][i]\*temp[i][k];

}

}

for(i = Filas - 1; i >= 0; i--, s = 0) {

for(j = i + 1; j < Filas; j++)

s += temp[i][j]\*x[j];

x[i] = (b[i] - s) / temp[i][i];

}

return x;

}

Considerar

N = Filas

|  |  |  |
| --- | --- | --- |
| *Constante* | *Frecuencia* | *Valor Frecuencia* |
| *C1* | *F1* | *(N-1)\*((N+1)/2)\*N* |
| *C2* | *F2* | *(N-1)\*((N+1)/2)\*N* |

T(N)<=C1\*F1+ C2\*F2

T(N)<=C1\*() +C2\*()

T(N)<=

Modelo de costo: 2

Función tilde: 2k

Orden de crecimiento: Cubico ()

1. multiplicaciones/divisiones

public double[] eliminacionGauss(double[] b) throws Exception {

if(Filas != Columnas || Filas != b.length) throw new Exception("La matriz debe ser NxN y el tamaño del vector debe ser N");

int k, i , j;

for(i=0; i < Filas; i++)

if(matriz[i][i] == 0) throw new Exception("La diagonal principal no puede contener ceros");

double s = 0;

double[] x = new double[Filas];

double[][] temp = matriz;

for(i = 0; i < Filas - 1; i++) {

for(j = i + 1; j < Filas; j++) {

b[j] = temp[i][i]\*b[j] - temp[j][i]\*b[i];

for(k = Filas - 1; k >= 0; k--)

temp[j][k] = temp[i][i]\*temp[j][k] - temp[j][i]\*temp[i][k];

}

}

for(i = Filas - 1; i >= 0; i--, s = 0) {

for(j = i + 1; j < Filas; j++)

s += temp[i][j]\*x[j];

x[i] = (b[i] - s) / temp[i][i];

}

return x;

}

Considerar

N = Filas

|  |  |  |
| --- | --- | --- |
| *Constante* | *Frecuencia* | *Valor Frecuencia* |
| *C1* | *F1* | *(N-1)\*((N+1)/2)\*N* |
| *C2* | *F2* | *(N-1)\*((N+1)/2)\*N* |

T(N)<=C1\*F1+ C2\*F2

T(N)<=C1\*() +C2\*(

T(N)<=

Modelo de costo: 2

Función tilde: 2k

Orden de crecimiento: Cubico ()