



# Estructuras de datos

*Arle Morales Ortiz*

*Docente*

*Ingeniería de software*



**PROGRAMA DE  
INGENIERÍA  
DE SOFTWARE**  
CUE AVH *Dual*

Las **matrices** tienen utilidad en una diversidad de campos, en especial en el **álgebra** lineal, especialmente para la resolución de **sistemas de ecuaciones lineales**.

Una matriz es una tabla de números ordenada en  $m$  filas y  $n$  columnas, cerrada entre paréntesis.

Cada elemento de la **matriz** se denomina  $a_{ij}$ . Y la matriz se representa por  $(a_{ij})$ , o, directamente por  $(A)$ . El subíndice  $i$  indica la fila y el subíndice  $j$ , la columna.

$$(a_{ij}) = A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

La matriz tiene  $m$  filas y  $n$  columnas. El primer elemento es  $a_{11}$  y el último  $a_{mn}$ .

La posición del elemento  $a_{12}$  es: primera fila y segunda columna.

En esta matriz están reflejadas las calificaciones obtenidas por los tres grupos de primero de un colegio.

	A	B	C (Grupos)
Sobresaliente	4	3	2
Satisfactorio	10	11	9
Básico	11	12	13
Insuficiente	6	5	7


# Matriz dispersa

Una matriz es un objeto de datos bidimensional que tiene  $m$  filas y  $n$  columnas, por lo tanto, un total de  $m*n$  valores. Si la mayoría de los valores de una matriz son 0, decimos que la matriz se llama **matriz dispersa**.

$$I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

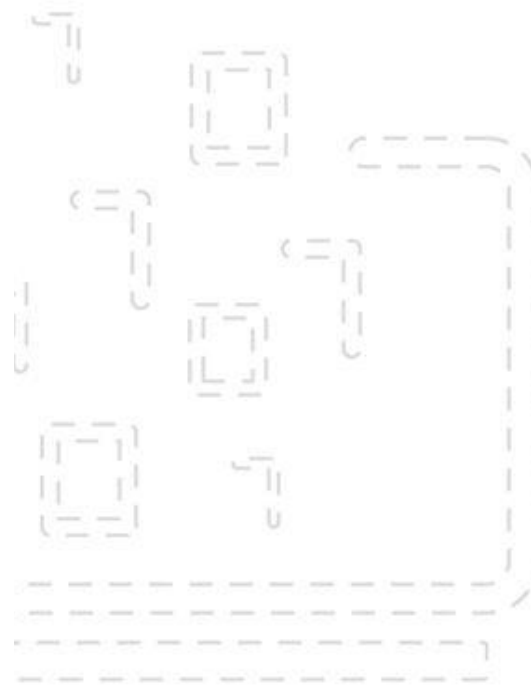
$$I_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Entonces, nuestro objetivo es contar el número de 0 presentes en la matriz. Si este recuento es mayor que  $(m * n)/2$ , imprima Sí, de lo contrario, No.

# Ejemplo 1

```
1 package co.cue.matriz.dispersa.clases;
2
3 public class Dispersa1 {
4     public static void main(String args[]) {
5         // Initialising and declaring
6         // variables and array
7         int array[][] = {{1, 0, 3}, {0, 0, 4}, {6, 0, 0}};
8
9         int m = 3;
10        int n = 3;
11        int counter = 0;
12
13        // Count number of zeros in the matrix
14
15        for (int i = 0; i < m; ++i)
16            for (int j = 0; j < n; ++j)
17                if (array[i][j] == 0)
18                    ++counter;
19        // Printing result
20
21        if (counter > ((m * n) / 2))
22            System.out.println("Yes");
23        else
24            System.out.println("No");
25    }
26 }
```



## ¿Por qué usar matriz dispersa en lugar de matriz simple?

- **Almacenamiento:** hay menos elementos distintos de cero que ceros y, por lo tanto, se puede usar menos memoria para almacenar solo esos elementos.
- **Tiempo de cálculo:** el tiempo de cálculo se puede ahorrar diseñando lógicamente una estructura de datos que atraviese solo elementos distintos de cero.

Conceptualmente, **la escasez** corresponde a sistemas con pocas **interacciones por pares**. Por ejemplo, considere una línea de bolas conectadas por resortes de una a la siguiente: este es un sistema escaso ya que solo se acoplan las bolas adyacentes. Por el contrario, si la misma línea de bolas tuviera resortes que conectan cada bola con todas las demás bolas, el sistema correspondería a una matriz densa.

El concepto de escasez es **útil en combinatoria y áreas de aplicación** como la teoría de redes y el análisis numérico, que normalmente tienen una baja densidad de datos o conexiones importantes. Las matrices dispersas grandes a menudo aparecen **en aplicaciones científicas o de ingeniería al resolver ecuaciones diferenciales parciales**.

La representación de una matriz dispersa mediante una matriz 2D conduce al desperdicio de mucha memoria, ya que los ceros en la matriz no sirven en la mayoría de los casos. Entonces, en lugar de almacenar ceros con elementos distintos de cero, solo almacenamos elementos distintos de cero. Esto significa almacenar elementos distintos de cero con triples (fila, columna, valor).


Las representaciones de matrices dispersas se pueden hacer de muchas maneras, a continuación se muestran dos representaciones comunes:

- ❖ representación de matriz
- ❖ Representación de lista enlazada

### Método 1: uso de matrices:

la matriz 2D se usa para representar una matriz dispersa en la que hay tres filas nombradas como

- **Fila:** índice de la fila, donde se encuentra el elemento distinto de cero
- **Columna:** índice de la columna, donde se encuentra el elemento distinto de cero
- **Valor:** valor del elemento distinto de cero ubicado en el índice - (fila, columna)



0	0	3	0	4
0	0	5	7	0
0	0	0	0	0
0	2	6	0	0



Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6



# Ejemplo:

```

1 package co.cue.matriz.dispersa.clases;
2
3 public class Dispersa2 {
4     public static void main(String[] args) {
5         int sparseMatrix[][]
6             = {
7             {0, 0, 3, 0, 4},
8             {0, 0, 5, 7, 0},
9             {0, 0, 0, 0, 0},
10            {0, 2, 6, 0, 0}
11        };
12
13        int size = 0;
14        for (int i = 0; i < 4; i++) {
15            for (int j = 0; j < 5; j++) {
16                if (sparseMatrix[i][j] != 0) {
17                    size++;
18                }
19            }
20        }
21
22        // number of columns in compactMatrix (size) must be
23        // equal to number of non - zero elements in
24        // sparseMatrix
25        int compactMatrix[][] = new int[3][size];
26
27        // Making of new matrix

```

parte 1

```

27        int k = 0;
28        for (int i = 0; i < 4; i++) {
29            for (int j = 0; j < 5; j++) {
30                if (sparseMatrix[i][j] != 0) {
31                    compactMatrix[0][k] = i;
32                    compactMatrix[1][k] = j;
33                    compactMatrix[2][k] = sparseMatrix[i][j];
34                    k++;
35                }
36            }
37        }
38
39        for (int i = 0; i < 3; i++) {
40            for (int j = 0; j < size; j++) {
41                System.out.printf("%d ", compactMatrix[i][j]);
42            }
43            System.out.printf("\n");
44        }
45    }
46 }

```

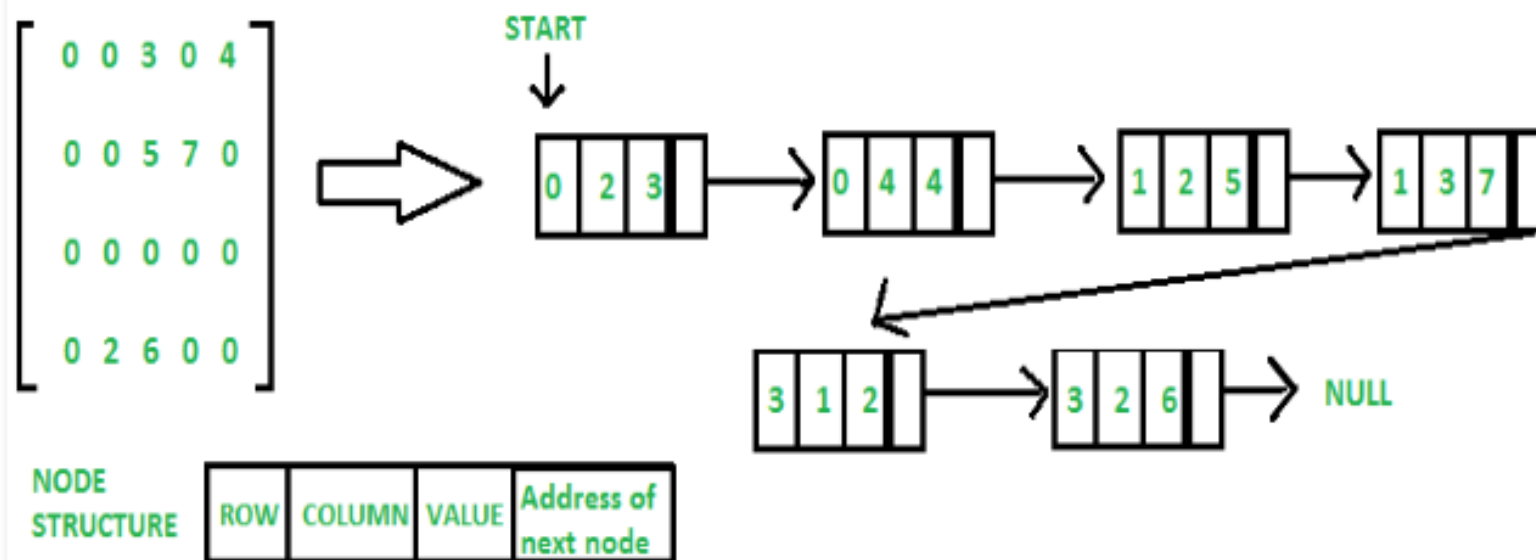
parte 2



## Método 2: uso de listas

vinculadas En la lista vinculada, cada nodo tiene cuatro campos. Estos cuatro campos se definen como:

- **Fila:** índice de la fila, donde se encuentra el elemento distinto de cero
- **Columna:** índice de la columna, donde se encuentra el elemento distinto de cero
- **Valor:** valor del elemento distinto de cero ubicado en el índice - (fila, columna)
- **Nodo siguiente:** Dirección del nodo siguiente



## Ejemplo:

```
1 package co.cue.matriz.dispersa.clases;
2
3 public class Node {
4     public int row;
5     public int col;
6     public int value;
7     public Node next;
8
9     public Node(int r, int c, int val)
10    { row = r; col = c; this.value = val; }
11 }
```

```
[ ... ]
```

```

1 package co.cue.matriz.dispersa.principal;
2
3 import co.cue.matriz.dispersa.clases.Node;
4
5 public class Principalnode {
6     public static void main(String[] args)
7     {
8         /*Assume a 4x4 sparse matrix */
9         int sparseMatrix[][] = {
10             {0, 0, 1, 2},
11             {3, 0, 0, 0},
12             {0, 4, 5, 0},
13             {0, 6, 0, 0}
14         };
15         Node start = null; /*Start with the empty list*/
16         Node tail = null;
17         int k = 0;
18         for (int i = 0; i < 4; i++)
19             for (int j = 0; j < 4; j++)
20             {
21                 if (sparseMatrix[i][j] != 0) /*Pass only non-zero values*/
22                 {
23                     Node temp = new Node(i, j, sparseMatrix[i][j]);
24                     temp.next = null;
25                     if(start == null){
26                         start = temp;
27                         tail=temp;

```

```

28
29
30
31
32
33
34
35
36
37
38
39
40
41

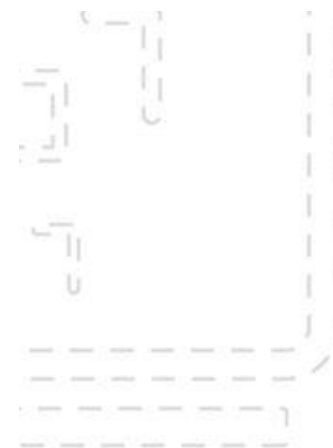
```

```

    }
    else{
        tail.next = temp;
        tail = tail.next;
    }
}

Node itr = start;
while(start != null){
    System.out.println(start.row + " " + start.col + " " + start.value);
    start = start.next;
}
}

```



resultado

```

0 2 1
0 3 2
1 0 3
2 1 4
2 2 5
3 1 6

```

# Ejercicio 3:

1. Hacer una simulación del juego buscaminas usando una matriz dispersa.

*El algoritmo debe como mínimo tener una matriz 4 X 4*

*Decir aciertos y cuando se pisa una mina.*



# ¡Gracias!

*Arle Morales Ortiz*  
2022

