

MATRICES DISPERSAS.

Presentado por:

**Carlos Andrés Cuervo 1625639.
carlos.cuervo@correounivalle.edu.co**

**Juan David Tello 1628950.
tello.juan@correounivalle.edu.co**

**Christian David Millan 1628953.
christian.millan@correounivalle.edu.co**

**Docente:
Ing. Carlos A. Delgado.**

**Escuela Ingeniería de Sistemas.
Universidad del valle.
F.A.D.A
Junio/2018.**

Aseguramos que nuestro grupo NO ha copiado de nadie, ni dado copia a nadie, la solución que a continuación presentamos

GestionDeArchivos
<pre>-matriz: int[][] -filas: int -columnas: int +GestionDeArchivos() +getRoute(): String +loadMatriz(route:String): void +getFilas(): int +getColumnas(): int +getMatriz(): int[][]</pre>

CSR
<pre>-matriz: int[][] -matrizSum: int[][] -listaValores: ArrayList<Integer> -listaFilas: ArrayList<Integer> -listaCFilas: ArrayList<Integer> -listaColumnas: ArrayList<Integer> -arrayFila: int[] -arrayColumna: int[] -filas: int -columnas: int +CSR() +loadMatriz(matrizFile:int[][], fila:int, columna:int): void +loadMatrizSum(matrizFile:int[][], fila:int, columna:int): void +csrFormat(): void +showMatriz(): void +showMatriz2(matriz:int[][]): void +showArrayList(lista:ArrayList<Integer>): void +fillMatriz(matrizAux:int[][]): void +generateRepresentation(archivo:File, escribir:FileWriter): void +createFileMatriz(route:String): void +getElement(fila:int, columna:int): void +sumMatrices(route:String): void +transposedMatriz(route:String): void +multiplicador(fila:int[], columna:int[]): int +squareMatriz(route:String): void +setPosition(route:String, fila:int, columna:int, numero:int): void +getRowAux(fila:int): void +getRow(fila:int): void +getColumn(columna:int): void +getColumnAux(columna:int): void +createFileRepresentation(route:String): void</pre>

Coordenado
<pre>-matriz: int[][] -matrizSum: int[][] -pilaValores: Stack<Integer> -pilaFilas: Stack<Integer> -pilaColumnas: Stack<Integer> -pilaValoresSum: Stack<Integer> -pilaFilasSum: Stack<Integer> -pilaColumnasSum: Stack<Integer> -listaValores: ArrayList<Integer> -listaFilas: ArrayList<Integer> -listaColumnas: ArrayList<Integer> -arrayFila: int[] -arrayColumna: int[] -filas: int -columnas: int +Coordenado() +loadMatriz(matrizFile:int[][], fila:int, columna:int): void +loadMatrizSum(matrizFile:int[][], fila:int, columna:int): void +coordinateFormat(): void +coordinateFormatSum(): void +showMatriz(): void +showArray(lista:ArrayList<Integer>): void +showMatriz2(matriz:int[][]): void +showStack(pila:Stack<Integer>): void +fillArrayList(arreglo:int[], pila:Stack<Integer>): void +generateRepresentation(archivo:File, escribir:FileWriter): void +cleanStack(pila:Stack<Integer>): void +sumMatrices(route:String): void +transposedMatriz(route:String): void +multiplicador(fila:int[], columna:int[]): int +squareMatriz(route:String): void +setPosition(route:String, fila:int, columna:int, numero:int): void +getRowAux(fila:int): void +getRow(fila:int): void +getColumn(columna:int): void +getColumnAux(columna:int): void +getElement(fila:int, columna:int): void +createFileMatriz(route:String): void +createFileRepresentation(route:String): void</pre>

CSC

```

- matriz: String [ ][ ]
- Tmatriz: String [ ][ ]
- m: String [ ][ ]
- tamañofila: int
- tamañocolumna: int
- Listavaloresx: ArrayList<String>
- ListaFilas: ArrayList<String>
- Cabezas: ArrayList<String>
- Vacías: ArrayList<Integer>
- Listacolumnas: ArrayList<Integer>
- CCOLUMNAS: ArrayList<Integer>
- nuevoVALORES: ArrayList<String>
- nuevoFILAS: ArrayList<Integer>
- nuevoCCOLUMNAS: ArrayList<Integer>
- CCOLUMNAS2: ArrayList<Integer>
- columnasX: int [ ][ ]
- filasX: int [ ][ ]

+ CSC ()
+ CSCFormat(): void
+ TRANSPUESTA(route: String): void
+ ImprimeMATRIZaTEXTO(route: String): void
+ ImprimeColumna(): void
+ ImprimeFila(): void
+ ImprimeElemento(): void
+ imprimeMODIFICARPOSICION(route: String): void
+ getListavaloresx(): ArrayList<String>
+ getListafilas(): ArrayList<String>
+ getCabezas(): ArrayList<String>
+ getVacías(): ArrayList<Integer>
+ getCCOLUMNAS(): ArrayList<Integer>
+ loadMatriz(matrizFile: int[ ][ ], fila: int, columna: int): void
+ showMatriz(): void
+ CCOLUMNAS(x: String[ ][ ]): void
+ Valores(y: String[ ][ ]): void
+ Filas(y: String[ ][ ]): void
+ ObtenerMatriz(valores: ArrayList<String>, filas:
ArrayList<String>, ccolumnas: ArrayList<Integer>): void
+ ObtieneColumna(valores: ArrayList<String>, filas1:
ArrayList<String>,ccolumnas: ArrayList<Integer>,Ncolumna:
String): void
+ ObtenerElemento(valores: ArrayList<String>, filas1
ArrayList<String>, ccolumnas: ArrayList<Integer>,Nfila: String ,CC
: String): void
+ ObtenerFila(valores: ArrayList<String>,filas1: ArrayList<String>
,ccolumnas: ArrayList<Integer>,Nfila: String): void
+ ModificaELEMENTO(fila: int, columna: int, elemento: String)
+ createFileRepresentation(route: String): void
+ generateRepresentation(archivo: File, escribir: FileWriter): void
+ MATRIZATXT(matriz: String[ ][ ],route: String): void
+ createFileRepresentation2(route: String): void
+ multiplicador(fila: int[ ], columna: int[ ]): int
+ ObtieneColumnaAuxiliar(Ncolumna: String): void
+ ObtenerFilaAuxiliar(Nfila: String): void
+ MatrizCuadrada(String route): void
+ ObtenerMatrizT(valores: ArrayList<String>, filas:
ArrayList<String>, ccolumnas: ArrayList<Integer>, route: String):
void

```

FORMATO COORDENADO

Explicación de la representación:

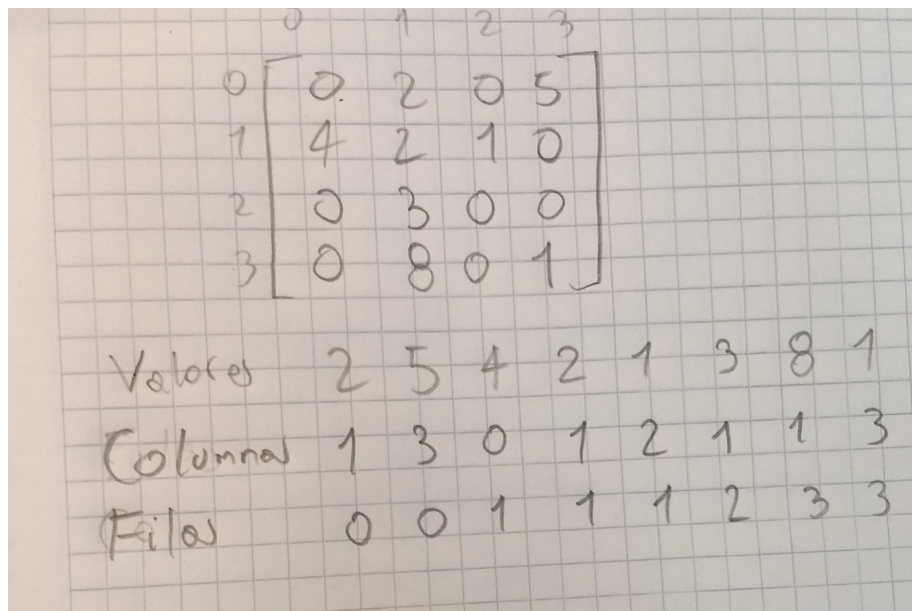
Primero leemos la cantidad de filas y columnas, que son los dos primeros valores del .txt, creamos una matriz del tamaño que se leyó para almacenar todos los valores del txt.

Con la matriz ya leída ahora pasamos los valores de filas, columnas, y valores de determinada posición a 3 distintas pilas, esto se hace recorriendo la matriz con un for anidado.

```
for(int i=0; i<filas; i++){  
    for(int j=0; j<columnas; j++){  
  
        if((matriz[i][j]==0)==false){  
  
            pilaValores.push(matriz[i][j]);  
            pilaFilas.push(i);  
            pilaColumnas.push(j);  
        }  
    }  
}
```

Ya teniendo las pilas con los valores correspondientes, para mostrarlas, simplemente se van sacando los valores recorriendo las pilas.

Ejemplo manual:



The image shows a handwritten example on graph paper. At the top, a 4x4 matrix is written with row indices 0-3 on the left and column indices 0-3 on top. The matrix contains the following values:

	0	1	2	3
0	0	2	0	5
1	4	2	1	0
2	0	3	0	0
3	0	8	0	1

Below the matrix, three rows of values are written, representing the contents of three stacks (pilas):

Valores	2	5	4	2	1	3	8	1
Columnas	1	3	0	1	2	1	1	3
Filas	0	0	1	1	1	2	3	3

Estrategia:

Se utilizaron dos tipos de estructuras para lograr todas las operaciones, pila, arreglos y listas la estrategia que se aplicó aquí fue recorrer todos los valores de las 3 estructuras que me representan valores, columnas y filas, como en esta representación el índice de fila y columna me dan la posición del valor en la matriz fue muy sencillo y mecánico realizar las operaciones.

Funciones implementadas:

loadMatriz(int[][] matrizFile,int fila, int columna)

Recibe la matriz de la clase GestionDeArchivos y la carga a la matriz de la clase Coordinate.

loadMatrizSum(int[][] matrizFile,int fila, int columna)

Recibe la matriz de la clase GestionDeArchivos que será sumada con la cargada inicialmente.

coordinateFormatSum()

Llena las pilas de la representación de la matriz que sumará la inicial con los valores correspondientes.

sumMatrices(String route)

Suma las matrices.

coordinateFormat()

Llenas la pilas, pilaValores,PilaColumnas,PilaFilas con los valores correspondientes para la representación coordinada.

showMatriz()

Muestra la matriz leída en consola.

showMatriz2(int[][] matriz)

Muestra una matriz dada en consola.

showArray(ArrayList<Integer> lista)

Dada una lista la imprime en pantalla.

showStack(Stack<Integer> pila)

Muestra una pila dada en consola.

fillArrayList(int[] arreglo,Stack<Integer> pila)

Dada una pila y un arreglo, saca los valores de la pila y los pasa al arreglo.

generateRepresentation(File archivo, FileWriter escribir)

Con las 3 pilas que contienen la información de la matriz, lo que hace este método es recorrer las pilas imprimiendo en un documento de texto el contenido de estas.

cleanStack(Stack<Integer> pila)

Dada una pila, borra todo su contenido.

transposedMatriz(String route)

Genera un documento de texto con que retorna la matriz leída inicialmente de forma transpuesta, a su vez tambien anexa en el documento la representación coordenada.

multiplicador(int [] fila, int[] columna)

Recibe dos arreglos fila y columna y multiplica sus valores entre ellos para dar el resultado de determinada posición de la nueva matriz al cuadrado.

squareMatriz(String route)

Genera un documento de texto con que retorna la representación de la matriz leída inicialmente con el resultado de multiplicar la matriz con ella misma.

setPosition(String route,int fila,int columna,int numero)

Genera un documento de texto que retorna la matriz con la posición cambiada.

```
for(int i=0; i<=listaValores.length; i++){
    if(listaFilas[i]==fila && listaColumnas[i]==columna){
        listaValores[i]=numero;
        confirmn=true;
        break;
    }
}
```

getRow(int fila)

Muestra en pantalla todos los elementos que se encuentran presentes en una fila.

getRowAux(int fila)

Llena un arreglo con todos los elementos de una fila dada, esto es usado para la operación de elevar al cuadrado.

getColumn(int columna)

Muestra en pantalla todos los elementos que se encuentran presentes en una columna.

getColumnAux(int columna)

Llena un arreglo con todos los elementos de una columna dada, esto es usado para la operación de elevar al cuadrado.

getElement(int fila,int columna)

Dada una fila y columna muestra en pantalla el número que se encuentra en determinada posición.

createFileMatriz(String route)

Crea un documento de texto con la matriz que se obtiene a partir de la representación.

```
int aux=pilaValores.size();
for(int i=0; i<aux;i++){//Se colocan los valores de pilas valores en las :
    matrizFile[pilaFilas.peek()][pilaColumnas.peek()]=pilaValores.peek();
    pilaFilas.pop();
    pilaColumnas.pop();
    pilaValores.pop();
}
```

createFileRepresentation(String route)

Crea un documento de texto con los valores de las pilas de la representación coordenada.

COMPLEJIDAD

Crear representación: Una matriz tiene n filas y m columnas, para la representación en el formato coordenado utilizamos pilas, las cuales tienen complejidad $O(1)$, al insertar los datos en las pilas usamos dos ciclos anidados, por lo tanto la complejidad de esta operación sería $O(nm)$.

Obtener matriz: Partiendo de la representación coordenada, llenamos una matriz con los datos de las pilas, para llenar la matriz se utiliza un ciclo desde $i=0$ hasta $i<pilaValores.size()$, si la representación tiene n valores la complejidad de esta operación sería $O(n)$.

Obtener elemento: Para esta operación tenemos de igual forma el mismo ciclo que para obtener la matriz, $O(n)$.

Obtener fila: Para esta operación tenemos de igual forma el mismo ciclo que para obtener elemento, $O(n)$.

Obtener columna: Para esta operación tenemos de igual forma el mismo ciclo que para obtener fila, $O(n)$.

Modificar posición: Para esta operación tenemos de igual forma el mismo ciclo que para obtener fila, $O(n)$.

Elevar al cuadrado: El multiplicador tiene una complejidad de $O(n)$ en el peor de los casos

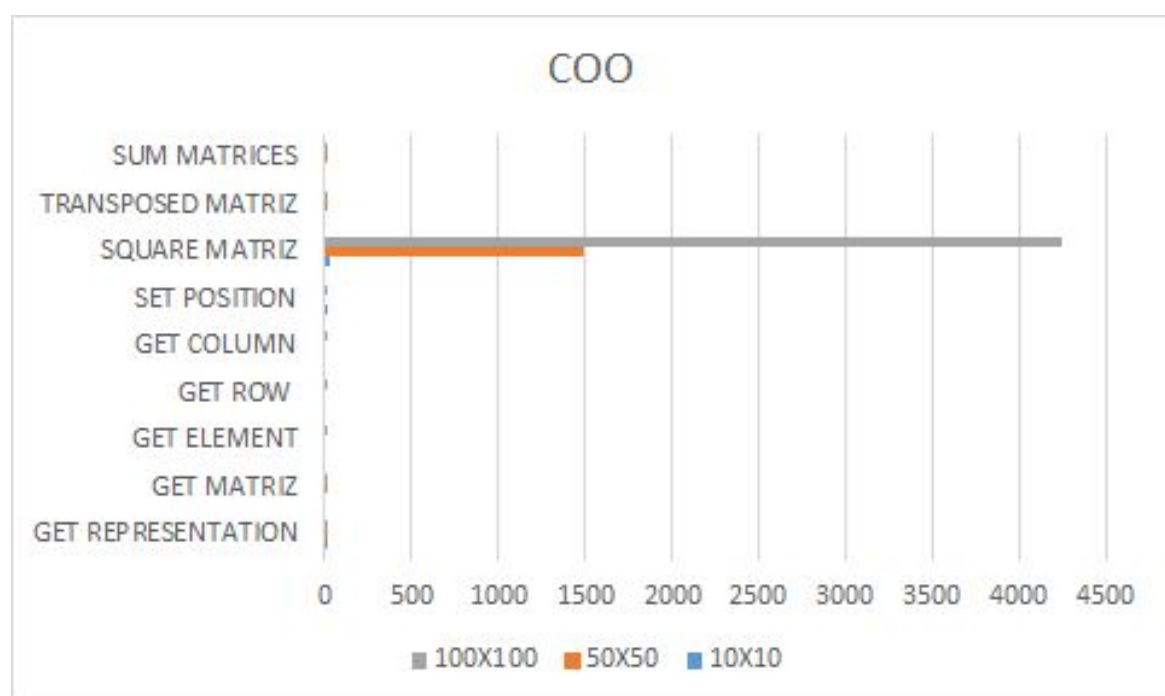
```
public static int multiplicador(int[] fila, int[] columna){
    int resultado=0;
    for(int i=fila.length-1; i>=0; i--){
        if(fila[i] !=0 && columna[i] !=0){
            resultado += fila[i]*columna[i];
        }
    }
    return resultado;
}
```

Anexando a ello esta operación del multiplicador se repite nm veces siendo n el tamaño de filas y m el tamaño de columnas, la complejidad total de la operación sería $O(n^2m)$

Matriz transpuesta: $O(nm)$, al igual que en la primer operación partimos de que una matriz tiene n filas y m columnas, y para transponer debemos recorrerla.

Suma de matrices: Dada la matriz inicial y la que pensamos sumar, solo es sumar estas dos en sus respectivas posiciones, $O(nm)$.

FORMATO COORDENADO				PRUEBA 1
OPERACIÓN	10X10	50X50	100X100	
GET REPRESENTATION	1	7	20	
GET MATRIZ	0	2	1	
GET ELEMENT	0	0	3	
GET ROW	0	0	1	
GET COLUMN	0	0	2	
SET POSITION	1	0	1	
SQUARE MATRIZ	30	1491	4239	
TRANPOSED MATRIZ	0	1	2	
SUM MATRICES	0	1	2	
TIEMPO EN MILISEGUNDOS				
FORMATO COORDENADO				PRUEBA 2
OPERACIÓN	10X10	50X50	100X100	
GET REPRESENTATION	0	9	22	
GET MATRIZ	0	1	3	
GET ELEMENT	0	0	2	
GET ROW	0	1	2	
GET COLUMN	0	0	1	
SET POSITION	0	1	1	
SQUARE MATRIZ	36	1520	3869	
TRANPOSED MATRIZ	1	3	3	
SUM MATRICES	0	2	2	
TIEMPO EN MILISEGUNDOS				



FORMATO COMPRIMIDO POR FILA

Explicación de la representación:

Para esta representación utilizamos 3 listas, partiendo de que la primer lista contiene los valores de la matriz de izquierda a derecha, la segunda contiene las columnas en las cuales se ubica un valor determinado y la última contiene la posición de los valores de los inicios de cada fila en la primer lista.

```
int contador=0;
for(int i=0; i<filas; i++){
    for(int j=0; j<columnas; j++){
        if((matriz[i][j]==0)==false){ //Si el valor de la
            listaValores.add(matriz[i][j]); //Agrego el valo
            listaColumnas.add(j); //Agrego la posicion de la
            contador++;
        }
        if(j==(columnas-1)){ //En la ultima iteracion
            listaCFilas.add(listaValores.size()-contador); //
        }
    }
    contador=0;
}
listaCFilas.add(listaCFilas.get(listaCFilas.size()-1)+1);
```

Agrego los valores de la matriz a la primer lista recorriendola normalmente al igual que la lista de columnas, sin embargo para última lista, agrego la posición en la cual está el primer valor de cada fila en la lista que contiene los valores.

Ejemplo manual:

	0	2	0	5				
	4	2	1	0				
	0	3	0	0				
	0	8	0	1				
V	2	5	4	2	1	3	8	1
C	1	3	0	1	2	1	1	3
CF	0	2	5	6	8			

Estrategia:

Para esta representación se utilizaron ArrayList, una vez obtenida la representación el problema a seguir fue como solucionar las demás operaciones, se noto que en la última lista, al restar un elemento con su anterior daba como resultado la cantidad de valores distintos de 0 que se presentaban en determinada fila, partiendo de esto, se construyó un ciclo anidado.

```
int aux=listaCFilas.size();
int contadorValores=0;
int contadorColumnas=0;
int contadorFilas=0;
int cantElementosFila=0;

for(int i=0; i<=aux;i++){
    if(!(contadorFilas==listaCFilas.size()-1)){//Si la lista de Punteros de inicio de filas no se ha recorrido pc
        cantElementosFila=(int) listaCFilas.get(contadorFilas+1)-(int) listaCFilas.get(contadorFilas);
        for(int j=0; j<cantElementosFila;j++){//Lleno la matriz por filas.
            matrizFile[contadorFilas][listaColumnas.get(contadorColumnas)]=listaValores.get(contadorValores);
            contadorValores++;
            contadorColumnas++;
        }
        contadorFilas++;
    }else{//Si el arreglo de punteros a inicios de fila se recorrio completamente
        //Lleno la ultima fila con los valores y columnas restantes
        matrizFile[contadorFilas-1][listaColumnas.get(contadorColumnas)]=listaValores.get(contadorValores);
        contadorValores++;
        contadorColumnas++;
    }
}
```

El cual se utilizó en todas las operaciones, haciendo breves modificaciones para lograr el objetivo de estas.

Funciones implementadas.

loadMatriz(int[][] matrizFile,int fila, int columna)

Carga la matriz leida en un atributo matriz principal.

loadMatrizSum(int[][] matrizFile,int fila, int columna)

Carga la matriz secundaria para ser sumada en un atributo matriz.

csrFormat()

Llena las 3 listas, listaValores, listaColumnas, listaCFilas, representando el formato comprimido por filas.

showMatriz()

Imprime en consola la matriz principal.

showMatriz2(int[][] matriz)

Dada una matriz la imprime en consola.

showArrayList(ArrayList<Integer> lista)

Dada una lista, la imprime en consola.

generateRepresentation(File archivo, FileWriter escribir)

Imprime en un archivo de texto las listas que representan el formato comprimido por fila.

fillMatriz(int[][] matrizAux)

Dada una matriz la llena con Ceros (0).

createFileMatriz(String route)

Crea una matriz a partir de la representación de formato coordinado.

getElement(int fila,int columna)

Dada una posición imprime en pantalla el valor de esta.

getColumn(int columna)

Dada una columna retorna todos los valores presentes en esta.

getColumnAux(int columna)

Dada una columna llena un arreglo con los valores de esta, este arreglo se utiliza en la operación elevar al cuadrado.

getRow(int fila)

Dada una fila retorna todos los valores presentes en esta.

getRowAux(int fila)

Dada una fila llena un arreglo con los valores de esta, este arreglo se utiliza en la operación elevar al cuadrado.

setPosition(String route,int fila, int columna,int numero)

Dada una posición y un valor, cambia el valor actual en la posición dada por el nuevo valor, muestra la nueva matriz en un archivo de texto.

multiplicador(int [] fila, int[] columna)

Recibe dos arreglos fila y columna y multiplica sus valores entre ellos para dar el resultado de determinada posición de la nueva matriz al cuadrado.

squareMatriz(String route)

Genera un documento de texto con que retorna la representación de la matriz leída inicialmente con el resultado de multiplicar la matriz con ella misma.

transposedMatriz(String route)

Imprime en un archivo de texto la matriz transpuesta.

sumMatrices(String route)

Dada un archivo de texto que contenga una matriz del mismo tamaño a la original, imprime en un archivo de texto, la suma de estas.

createFileRepresentation(String route)

Junto con generateRepresentation retorna la representación comprimida por filas en un archivo de texto.

COMPLEJIDAD.

Crear representación: Una matriz tiene n filas y m columnas, para la representación en el formato comprimido por filas utilizamos listas, las cuales tienen complejidad $O(n)$, al insertar los datos en las filas usamos dos ciclos anidados, por lo tanto la complejidad de esta operación sería $O(n \wedge 2m)$, por que para insertar un dato en una lista hay que recorrerla, y esto cuesta n .

Obtener matriz: Partiendo de la representación comprimida por filas, llenamos una matriz con los datos de las lista, para llenar la matriz se utiliza un ciclo desde $i=0$ hasta $i < \text{listaCFilas.size}()$, esta lista es de tamaño n al igual que la cantidad de filas y el ciclo interno se repite la cantidad de elemento que haya en una fila $O(nm)$, en el peor de los casos $O(n \wedge 2)$.

Obtener elemento: Para esta operación tenemos de igual forma el mismo ciclo que para obtener la matriz, $O(nm)$, en el peor de los casos $O(n \wedge 2)$.

Obtener fila: Para esta operación tenemos de igual forma el mismo ciclo que para obtener elemento, $O(nm)$, en el peor de los casos $O(n \wedge 2)$.

Obtener columna: Para esta operación sacamos el valor directo comparando posición a posición la lista de columnas con la de valores, $O(n)$.

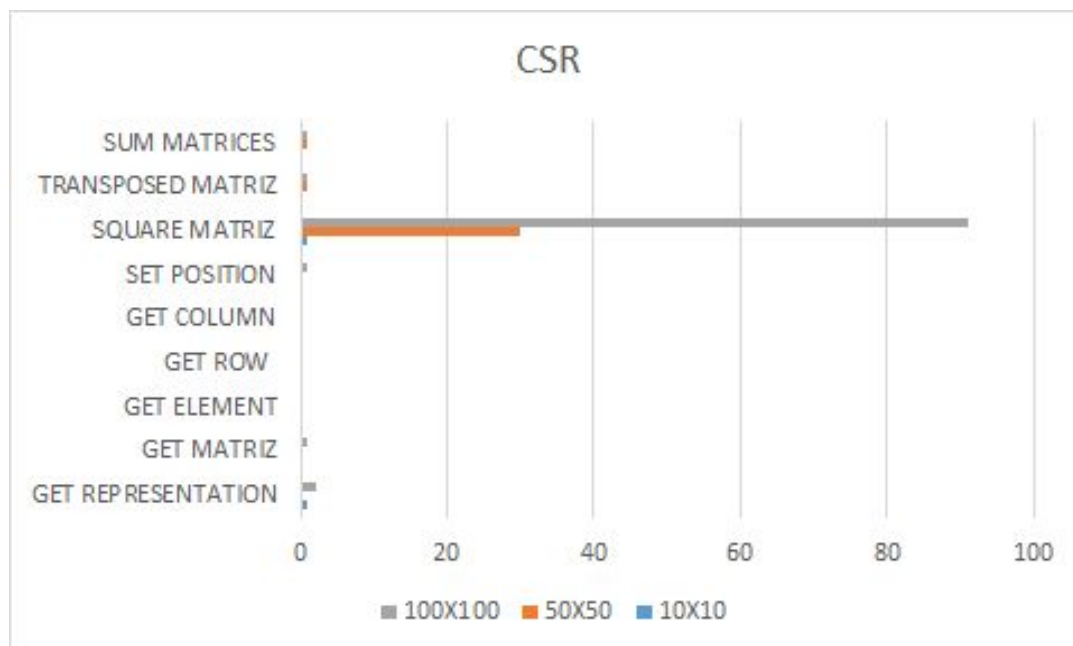
Modificar posición: Para esta operación tenemos de igual forma el mismo ciclo que para obtener fila, $O(nm)$, en el peor de los casos $O(n \wedge 2)$.

Elevar al cuadrado: Para esta operación tenemos de igual forma el mismo ciclo que para obtener la matriz, $O(nm)$, en el peor de los casos $O(n \wedge 2m)$.

Matriz transpuesta: Para esta operación tenemos de igual forma el mismo ciclo que para obtener la matriz, $O(nm)$, en el peor de los casos si las filas y columnas son de igual tamaño $O(n \wedge 2)$.

Suma de matrices: Dada la matriz inicial y la que pensamos sumar, solo es sumar estas dos en sus respectivas posiciones, $O(nm)$.

FORMATO CSR				PRUEBA 1
OPERACIÓN	10X10	50X50	100X100	
GET REPRESENTATION	1	0	2	
GET MATRIZ	0	0	1	
GET ELEMENT	0	0	0	
GET ROW	0	0	0	
GET COLUMN	0	0	0	
SET POSITION	0	0	1	
SQUARE MATRIZ	1	30	91	
TRANPOSED MATRIZ	0	1	1	
SUM MATRICES	0	1	1	
TIEMPO EN MILLISEGUNDOS				
FORMATO CSR				PRUEBA 2
OPERACIÓN	10X10	50X50	100X100	
GET REPRESENTATION	0	1	1	
GET MATRIZ	0	0	1	
GET ELEMENT	0	0	0	
GET ROW	0	0	0	
GET COLUMN	0	0	1	
SET POSITION	0	0	1	
SQUARE MATRIZ	0	27	57	
TRANPOSED MATRIZ	0	1	2	
SUM MATRICES	0	1	2	
TIEMPO EN MILLISEGUNDOS				



FORMATO COMPRIMIDO POR COLUMNA

Explicación de la representación:

Para esta representación utilizamos 3 listas, partiendo de que la primer lista contiene los valores de la matriz de arriba hacia abajo, la segunda contiene las filas en las cuales se ubica un valor determinado y la última contiene la posición de los valores de los inicios de cada columna en la primer lista.

```
248 public static void Valores(String[][] y){
249     //Saca valores de la matriz
250     for(int i=0; i<y[0].length;i++){
251         for(int j=0; j<y.length; j++){
252             if(y[j][i].equalsIgnoreCase("0")){}
253             else{
254                 String n = "";
255                 n = y[j][i];
256                 Listavaloresx.add(n);
257             }
258         }
259     }
260 }
261
262
263
264
265
266 static public void Filas(String[][] y){
267     //saca filas de la matriz
268     for(int i=0; i<y[0].length;i++){
269         for(int j=0; j<y.length; j++){
270             if(y[j][i].equalsIgnoreCase("0")){}
271             else{
272                 String n = "";
273                 n = Integer.toString(j);
274                 ListaFilas.add(n);
275             }
276         }
277     }
278 }
279
280
281
282
283
284 }
```



```

public static void CCOLUMNAS(String[][] x){
    //Saca las cabezas por el formato comprimido por columna de una matriz
    int[] prueba = new int[1];
    int cuenta = 0;
    int cuenta2 = 0;
    int cuenta3 = 0;
    int vaciallena=0;

    for(int i=0; i<x[0].length;i++){//recorre columnas

        for(int j=0; j<x.length; j++){//recorre filas

            if(x[j][i].equalsIgnoreCase("0")){cuenta3++; }//para saber si la c

            else{
                if(cuenta2<1){//entraca solo si encuentra una cabeza de la ma
                    Cabezas.add(x[j][i]);
                    if(prueba[0]==1){vaciallena=cuenta;//saca los valores de las c
                    prueba[0]=0;

                    CCOLUMNAS2.add(vaciallena);
                }
                vaciallena=0;
                CCOLUMNAS2.add(cuenta);
                cuenta2++;

                }cuenta++;

            }

        }

        if(cuenta3==matriz.length){//para saber cuando una columna es vacia y así
            //llenar un arreglo para luego con un condicional con el fin que en el
            prueba[0]=1;
        }else{}

        cuenta3=0;
        cuenta2=0;

        } CCOLUMNAS2.add(Listavaloresx.size());
    }
}

```

Agrego los valores de la matriz a la primer lista recorriendola de arriba hacia abajo al igual que la lista de filas, sin embargo para última lista, agrego la posición en la cual está el primer valor de cada columna en la lista que contiene los valores.

Ejemplo manual:

0	2	0	5
4	2	1	0
0	3	0	0
0	8	0	1

Valores: 4 2 2 3 8 1 5 1

Filas: 1 0 1 2 3 1 0 3

C.Columnas: 0 1 5 6 8

Estrategia:

Para esta representación se utilizaron ArrayList, una vez obtenida la representación el problema a seguir fue como solucionar las demás operaciones, se noto que en la última lista, al restar un elemento con su siguiente daba como resultado la cantidad de valores distintos de 0 que se presentaban en determinada columna, partiendo de esto, se construyó un ciclo anidado para cada operación, exceptuando el método "Obtener columna", pues para este sólo se usó un ciclo

Funciones implementadas.

loadMatriz(int[][] matrizFile,int fila, int columna)

Carga la matriz leida en un atributo matriz principal.

CSCFormat()

Contiene 3 metodos que Llenan las 3 listas, listaValores, listaFilas, listaCcolumnas, representando el formato comprimido por columnas.

showMatriz()

Imprime en consola la matriz principal.

generateRepresentation(File archivo, FileWriter escribir)

Con las 3 pilas que contienen la información de la matriz, lo que hace este método es recorrer las pilas imprimiendo en un documento de texto el contenido de estas.

createFileMatriz(String route)

Crea una matriz a partir de la representación de formato coordinado.

multiplicador(int [] fila, int[] columna)

Recibe dos arreglos fila y columna y multiplica sus valores entre ellos para dar el resultado de determinada posición de la nueva matriz al cuadrado.

ObtenerMatriz(ArrayList<String> valores, ArrayList<String> filas,ArrayList<Integer> ccolumnas)

Recibe la representación(3 listas con valores,filas y ccolumnas) y grafica la matriz guardandola en un archivo txt

ObtieneColumna(ArrayList<String>valores,ArrayList<String>filas1,ArrayList<Integer> > ccolumnas,String Ncolumna)

Con la representación y un número en este caso string que representa el número de la columna, imprime toda la columna de la matriz

ObtenerFila(ArrayList<String> valores, ArrayList<String> filas1,ArrayList<Integer> ccolumnas,String Nfila)

Con la representación de la matriz y un valor que indica la fila de la matriz que se quiere sacar, imprime toda la fila entera en consola

ObtenerElemento(ArrayList<String>valores,ArrayList<String>filas1,ArrayList<Integer> r> ccolumnas,String Nfila,String CC)

Con la representación, y un numero de filas, y uno de columna, se retorna el valor en la matriz

ModificaELEMENTO(int fila, int columna, String elemento)

Dados una posicion fila y una posicion columna, más el elemento a modificar, se modifica en la posición ingresada, con el elemento ingresado y se imprime en un txt con la representación ahora modificada

MATRIZATXT(String[][] matri,String route)

Método que imprime una matriz en un txt con parámetros una matriz y una ruta donde imprime

ObtieneColumnaAuxiliar(String numerocolumna)

Este método me guarda en un array estático una columna completa de la matriz según el número de columna ingresado

ObtenerFilaAuxiliar(String Nfila)

Recibe un número que me identifica la fila a imprimir, y la imprime en consola

MatrizCuadrada(String ruta)

A partir de una matriz la multiplica por ella misma para obtenerla cuadrada y la imprime en un txt en la ruta que tiene como parámetro

COMPLEJIDAD

Crear representación: Un método con el nombre de CSCFormat el cual contiene 3 métodos los cuales cada uno funcionan con una complejidad $O(n(m*n))$ la complejidad sale de primero, insertar en un arraylist el cual nos cuesta $O(n)$ esta complejidad es factor de la complejidad de recorrer 2 ciclos el primero recorre el tamaño de columnas de la matriz ósea m y el segundo anidado recorre el tamaño de las filas de la matriz ósea n .

ObtenerMatriz: Este método recorre con un ciclo el tamaño de columnas de la matriz y otro ciclo interno recorre el tamaño de las filas de la matriz, insertando valores en un arraylist que tiene por complejidad $O(n)$, el resultado de la complejidad por lo tanto es $O(n(m*n))$

ObtieneColumna: Este método tiene complejidad $O(n)$ porque solo usa un ciclo el cual recorre únicamente el tamaño de las filas de matriz para ir llenando la columna como tal, ya que el método tiene como parámetro la posición de la columna entonces esta posición trabaja como un puntero para localizar la columna de forma directa

ObtenerElemento: Este método tiene un ciclo el cual recorre el tamaño de las columnas-1 y otro ciclo interno que recorre los elementos de la columna actual con los elemento de la columna, e imprime el elemento especificado, por lo tanto la complejidad es $O(n^2)$

ObtenerFila: Este método tiene un ciclo el cual recorre el tamaño de las columnas-1 y otro ciclo interno que recorre los elementos de la columna actual con los elemento de la columna, y agrega el elemento a un nuevo arrayList si su fila coincide con la especificada; luego imprime este último arrayList, por lo tanto la complejidad es $O(n^2)$

ModificarElemento: Este método recorre con un ciclo externo el tamaño de las columnas de la matriz y con un ciclo interno recorre el tamaño de las filas para modificar un elemento el cual se inserta en arraylist que tiene por complejidad para insertar $O(n)$ por lo tanto la complejidad total es de $O(n(m*n))$

MATRIZATXT: Este método su función es imprimir la matriz a partir de la representación contiene 2 ciclos uno externo y otro interno donde el externo recorre tamaño de columnas de la matriz y el interno recorre tamaño de las filas de la matriz teniendo por tanto complejidad $O(n*m)$

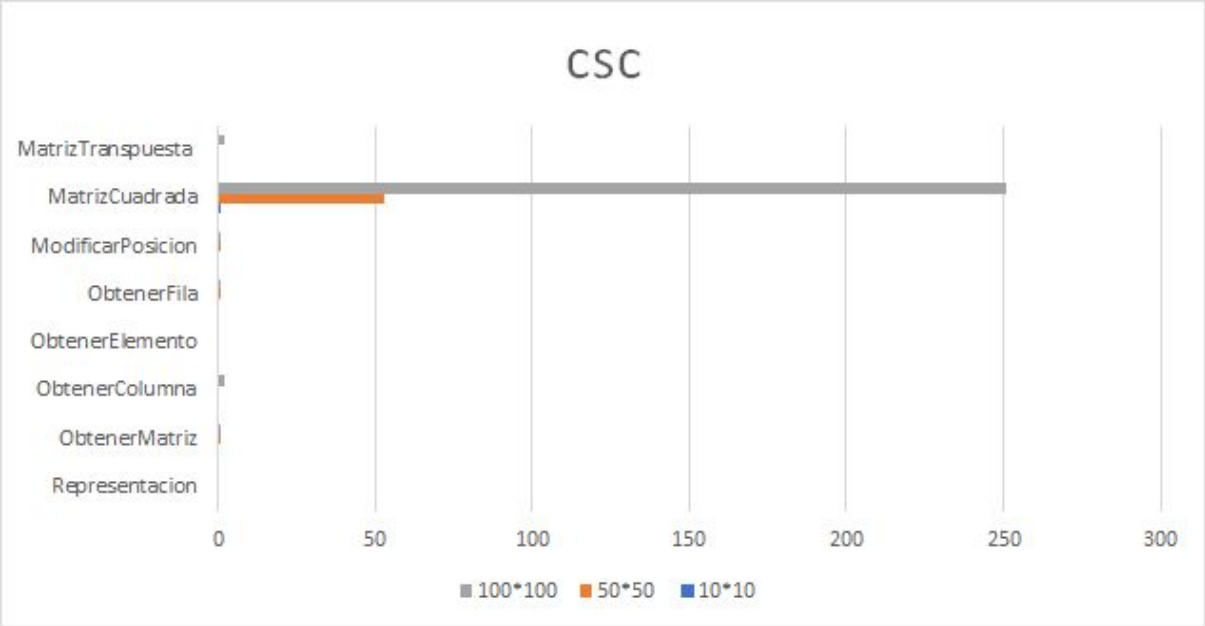
ObtieneColumnaAuxiliar: Este método es muy similar al descrito anteriormente ObtieneColumna solo que en este concretamente se tiene como parámetro el un numero de columna, su complejidad es $O(n)$ por tener un solo ciclo el cual recorre el tamaño de las filas de la matriz

ObtenerFilaAuxiliar: Este método similar a ObtenerFila; se diferencia en que los elementos que coinciden con la fila especificada se guardan en un arreglo estático y este último no se imprime, por lo tanto tiene la misma complejidad $O(n^2)$

MatrizCuadrada: Este método contiene 2 ciclos uno externo que recorre el tamaño de las columnas y uno interno que recorre el tamaño de las filas, además de insertar en arraylist elemento, por lo tanto la complejidad es $O(n(m*n))$ de insertar * recorrer tamaños de columnas y filas

MatrizTranspuesta: Este método recibe como parámetro tres arreglos con valores, filas y ccolumnas, después pasa a representarlos en una matriz la cual es la matriztranspuesta de la representación que entra como parametro, usa un ciclo externo que me recorre tamañoocolumnas y uno interno que recorre tamañoofilas asignando en un array doble que tiene complejidad $O(1)$ por lo tanto la complejidad total es de $O(n*m)$

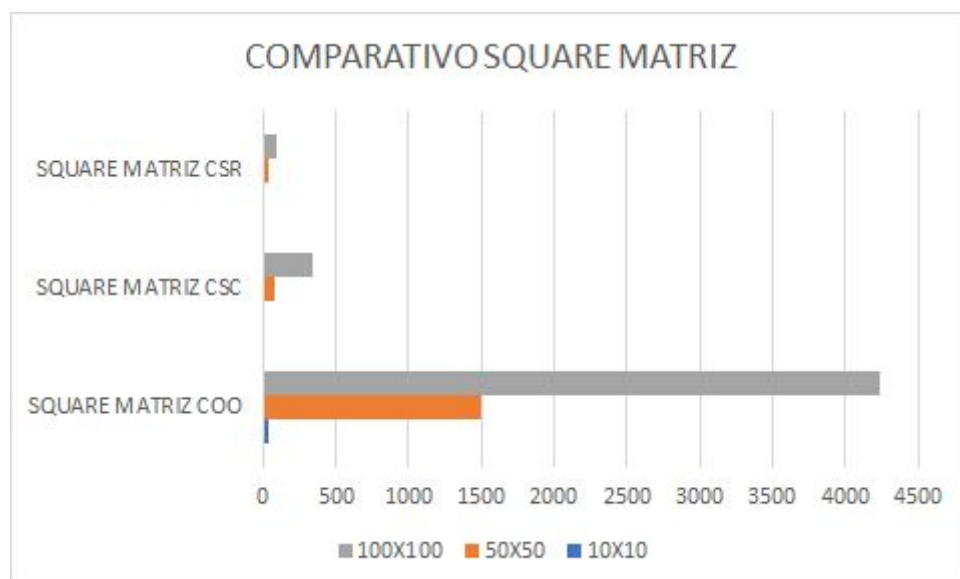
		FORMATO CSC		PRUEBA1
OPERACION	10*10	50*50	100*100	
Representacion	0	0	0	
ObtenerMatriz	0	1	1	
ObtenerColumna	0	0	2	
ObtenerElemento	0	0	0	
ObtenerFila	0	1	1	
ModificarPosicion	0	1	1	
MatrizCuadrada	1	72	340	
MatrizTranspuesta	0	0	2	
	TIEMPO EN MILLISEGUNDOS			
		FORMATO CSC		PRUEBA2
OPERACION	10*10	50*50	100*100	
Representacion	0	0	0	
ObtenerMatriz	0	1	1	
ObtenerColumna	0	1	2	
ObtenerElemento	0	0	0	
ObtenerFila	0	1	1	
ModificarPosicion	0	1	1	
MatrizCuadrada	0	53	251	
MatrizTranspuesta	0	0	0	
	TIEMPO EN MILLISEGUNDOS			



ANÁLISIS COMPARATIVO.

Para efectos prácticos y visuales se hace una tabla de comparación para la función SQUARE MATRIZ de cada una de las representaciones.

COMPARATIVO SQUARE MATRIZ EN LAS TRES REPRESENTACIONES			
OPERACIÓN	10X10	50X50	100X100
SQUARE MATRIZ COO	30	1491	4239
SQUARE MATRIZ CSC	1	72	340
SQUARE MATRIZ CSR	1	30	91



CONCLUSIONES.

Teniendo en cuenta que la complejidad teórica para SQUARE MATRIZ en cada una de las representaciones es $O(n^2m)$ se puede observar una notable diferencia en la complejidad práctica entre la representación **coordenada** y las representaciones **comprimada por filas** y **comprimada por columnas**, cuando la entrada es una matriz de 50x50 en adelante, por lo tanto se puede inferir que usar representaciones comprimidas para trabajar una matriz es mucho menos costo, y por ende, más eficiente.

WEBGRAFÍA.

<https://www.youtube.com/watch?v=kzJ9Ux2xwSI>

<https://www.youtube.com/watch?v=Qi7FcjN7nsc&feature=youtu.be>

https://www.fing.edu.uy/inco/cursos/comp1/teorico/2011/clase_23_2011_matDispersas.pdf

http://bibing.us.es/proyectos/abreproy/11926/fichero/Tercera+parte+CODIGO+del+PROYECTO%252FIV_MATRICES+DISPERSAS.pdf