## Línea horizontal



MATRICES DISPERSAS

Proyecto del curso Fundamentos de análisis y diseño de algoritmos.

**─**

Jeisson Lasso (201367647)

Pedro Pascuas (201552002)

Universidad del Valle - Sede Norte del Cauca

# 

[**Introducción**](#_au51mny0sx6) **2**

[**Comparación de los algoritmos**](#_4p7xi5bvhxdr) **3**

[Formato coordenada (FC)](#_56kfpodyq5td) 3

[Formato de fila dispersa comprimida (CSR)](#_ucse0oith95a) 4

[Formato de columna dispersa comprimida (CSC)](#_yse1hnkyjse6) 5

[**Explicación y ejemplos por cada algoritmo**](#_bnidp1wcm2vx) **6**

[Explicación algoritmo: Formato coordenada (FC)](#_szzis2kx6bu) 6

[Ejemplo:](#_daicr1tst1af) 6

[Explicación algoritmo: Formato de fila dispersa comprimida (CSR)](#_rl1iv9hwbvr5) 7

[Ejemplo:](#_t9z3iw18ex5h) 7

[Explicación algoritmo: Formato de columna dispersa comprimida (CSC)](#_ot5sizun0y9) 8

[Ejemplo:](#_n0sh68g5cfa4) 8

# 

# Introducción

En este documento planteamos los aspectos fundamentales del proyecto del curso Fundamentos de análisis y diseño de algoritmos, como la comparación de los algoritmos Formato Coordenada (FC), Formato de columna dispersa comprimida (CSC) y Formato de fila dispersa comprimida (CSR); utilizados para obtener la representación comprimida de matrices dispersas. También se explica el proceso de cada algoritmo para obtener la representación comprimida de las matrices dispersas; etc.

# 

# Comparación de los algoritmos

## Formato coordenada (FC)

Este algoritmo fue el más fácil de implementar, pero al parecer, por lo general, la simplicidad es una característica antagónica de la efectividad. Ya que podemos concluir que este algoritmo es el que menos comprime las matrices dispersas, ya que siempre crea 3 vectores, todos de tamaño n, en donde n es la cantidad de valores no nulos y diferentes de cero (0), a diferencia de los otros algoritmos, en los cuales el tercer vector no siempre es de tamaño n; al contrario el tamaño del tercer vector es menor a n.

El tiempo de ejecución se estima en términos de la cantidad de instrucciones  
que el algoritmo debe ejecutar y una constante que depende de las características de hardware de la máquina, de la forma en que implementamos este algoritmo, la cantidad de instrucciones está dada por:

**6 \* j \* i + 2**

En donde “j” es el número de columnas e “i” es el número de filas, 6 es el número de instrucciones que se ejecutan en el ciclo más interno y 2 es el número de instrucciones que se ejecutan cuando falla la condición de los 2 ciclos utilizados.

## 

## Formato de fila dispersa comprimida (CSR)

Aquí ya se aumenta la complejidad de implementación considerablemente, ya que en el tercer vector, en lugar de simplemente almacenar los índices de las columnas de los valores no nulos y diferentes de ceros debemos almacenar, los índices de **donde empiezan** los valores de cada fila, pero no los índices de la matriz dispersa, sino los índices del segundo vector, el cual almacena los índices de la columna de los valores no nulos y diferentes de cero (0) de la matriz dispersa. Esto es una gran mejora en cuanto efectividad, ya que la representación es más pequeña que la representación por formato coordenado. Básicamente la idea de este algoritmo es : “No es necesario almacenar los índices específicos de la fila de cada valor, porque tenemos los índices específicos de cada columna y el tercer vector mejor lo utilizamos para saber los intervalos de valores del primer vector que van en cada fila ”

El tiempo de ejecución se estima en términos de la cantidad de instrucciones  
que el algoritmo debe ejecutar y una constante que depende de las características de hardware de la máquina, de la forma en que implementamos este algoritmo, la cantidad de instrucciones está dada por:

**10 \* j \* i + i + 2**

En donde “j” es el número de columnas e “i” es el número de filas, 10 es el número de instrucciones que se ejecutan en el ciclo más interno y 2 es el número de instrucciones que se ejecutan cuando falla la condición de los 2 ciclos utilizados. Además se le suma “i” porque en el ciclo externo siempre se ejecuta una operación.

## Formato de columna dispersa comprimida (CSC)

Este algoritmo tiene la misma complejidad de implementación que el algoritmo CSR, ya que también en el tercer vector, en lugar de simplemente almacenar los índices de los valores no nulos y diferentes de ceros debemos almacenar, los índices de **donde empiezan** los valores de cada columna, pero no los índices de la matriz dispersa, sino los índices del segundo vector, el cual almacena los índices de la fila de los valores no nulos y diferentes de cero (0) de la matriz dispersa. Esto es una gran mejora en cuanto efectividad, ya que la representación es más pequeña que la representación por formato coordenado. Básicamente la idea de este algoritmo es : “No es necesario almacenar los índices específicos de la columna de cada valor, porque tenemos los índices específicos de cada fila y el tercer vector mejor lo utilizamos para saber los intervalos de valores del primer vector que van en cada columna ”

El tiempo de ejecución se estima en términos de la cantidad de instrucciones  
que el algoritmo debe ejecutar y una constante que depende de las características de hardware de la máquina, de la forma en que implementamos este algoritmo, la cantidad de instrucciones está dada por:

**10 \* j \* i + i + 2**

En donde “j” es el número de columnas e “i” es el número de filas, 10 es el número de instrucciones que se ejecutan en el ciclo más interno y 2 es el número de instrucciones que se ejecutan cuando falla la condición de los 2 ciclos utilizados. Además se le suma “i” porque en el ciclo externo siempre se ejecuta una operación.

# 

# Explicación y ejemplos por cada algoritmo

## Explicación algoritmo: Formato coordenada (FC)

for (int i = 0; i < matrizCompleta.size(); i++)

for (int j = 0; j < matrizCompleta.get(i).size(); j++) {

if( matrizCompleta.get(i).get(j) != 0 ){

valores.add( matrizCompleta.get(i).get(j) );

indicesFila.add( i );

indicesColumna.add( j );

}   
}

La idea de este algoritmo es simple, consiste en recorrer la matriz completa y cada vez que encontramos un valor no nulo y diferente de cero, lo almacenamos en el vector “valores” y también almacenamos su posición de fila en el vector “indicesFila” y su posición de columna en “indicesColumna”.

### Ejemplo:

Matriz dispersa completa:   
73 0 27  
0 97 96  
75 91 73  
  
La representación en formato coordenado es:   
Valores: 73,27,97,96,75,91,73  
indicesFila: 0,0,1,1,2,2,2  
indicesColumna: 0,2,1,2,0,1,2

## 

## Explicación algoritmo: Formato de fila dispersa comprimida (CSR)

for (int i = 0, contadorIndicesColumnas = 0; i < matrizCompleta.size(); i++) {  
 primerValorDeLafila = true;  
 for (int j = 0; j < matrizCompleta.get(i).size(); j++) {  
 if( matrizCompleta.get(i).get(j) != 0 ){  
 valores.add( matrizCompleta.get(i).get(j) );  
 indicesColumnas.add( j );  
 if(primerValorDeLafila){  
 indicesEmpiezaValoresEnFila.add( contadorIndicesColumnas );  
 primerValorDeLafila = false;  
 }  
 contadorIndicesColumnas++;  
 }  
 }  
 }

La idea de este algoritmo es un poco más compleja que la del anterior, también se recorre toda la matriz y cada vez que encontramos un valor no nulo diferente de cero, guardamos dicho valor en el vector “valores” y su índice de columna en el vector “indicesColumnas” y comprobamos si dicho valor es el primero de su fila, si es así guardamos su índice con respecto al vector “indicesColumnas” en el vector “indicesEmpiezaValoresEnFila” el cual calculamos por un contador, llamado “contadorIndicesColumnas”, el cual no es lo mismo que “j”, porque no es el índice con respecto a la matriz completa dispersa, sino el índice con respecto al segundo vector.

### Ejemplo:

Matriz dispersa completa:   
0 0 2 0  
0 13 29 0  
60 0 0 0  
33 23 0 0  
  
La representación en formato comprimido por filas es:   
2,13,29,60,33,23  
2,1,2,0,0,1  
0,1,3,4

## Explicación algoritmo: Formato de columna dispersa comprimida (CSC)

for (int i = 0, contadorIndicesFilas = 0; i < matrizCompleta.get(0).size(); i++) {  
 primerValorDeLaColumna = true;  
 for (int j = 0; j < matrizCompleta.size(); j++) {  
 if( matrizCompleta.get(j).get(i) != 0 ){  
 valores.add( matrizCompleta.get(j).get(i) );  
 indicesFilas.add( j );  
 if(primerValorDeLaColumna){  
 indicesEmpiezaValoresEnColumna.add( contadorIndicesFilas );  
 primerValorDeLaColumna = false;  
 }  
 contadorIndicesFilas++;  
 }  
 }

En Este algoritmo también se recorre toda la matriz y cada vez que encontramos un valor no nulo diferente de cero, guardamos dicho valor en el vector “valores” y su índice de columna en el vector “indicesFilas” y comprobamos si dicho valor es el primero de su columna, si es así guardamos su índice con respecto al vector “indicesFilas” en el vector “indicesEmpiezaValoresEnColumna” el cual calculamos por un contador, llamado “contadorIndicesFilas”, el cual no es lo mismo que “i”, porque no es el índice con respecto a la matriz completa dispersa, sino el índice con respecto al segundo vector.

### Ejemplo:

Matriz dispersa completa:   
60 40 71 0 33  
84 91 0 0 0  
0 0 6 0 63  
0 71 0 17 70  
94 33 75 0 0  
  
La representación en formato comprimido por columnas es:   
60,84,94,40,91,71,33,71,6,75,17,33,63,70  
0,1,4,0,1,3,4,0,2,4,3,0,2,3  
0,3,7,10,11