PROYECTO FINAL
DE MATRICES DISPERSAS

OSORIO PAZ CRISTIAN STEVEN
GARZON LOPEZ XAVIER
LOZANO ROJAS JHOAN SEBASTIAN

FUNDAMENTOS Y ESTRUCTURAS DE PROGRAMACION
GRUPO: A
DOCENTES: GERARDO SARRIA Y CARLOS RAMÍREZ

SANTIAGO DE CALI, MAYO 28 DE 2017

Desarrollo

1. Aspectos generales de cada implementación.

De forma general podemos considerar algunas acciones que realizamos durante todo el proyecto y se ven reflejadas en cada una de las tres implementaciones de las matrices dispersas. Primeramente, de las cosas que podemos considerar que permaneció constante es la forma en la que realizamos las pruebas a cada uno de los algoritmos respectivos a cada implementación y a partir de ello corroborábamos su veracidad; Dicho proceso, con fines de realizar una comprobación exhaustiva de nuestras funciones, decidimos realizar una función que genere matrices con entradas aleatorias, respetando la definición de matrices dispersas, permitiéndonos así poseer una mayor claridad en las operaciones.

Cabe resaltar que, en algunas funciones nuestro objetivo en primera instancia no fue buscar la mayor eficiencia en las soluciones que realizamos, más que todo nos centramos en que la solución que planteáramos funcionara en todos los casos posibles y sin errores.

Ya centrándonos en cada implementación, podemos considerar y explicar las siguientes decisiones tomadas para la representación de las listas y por ende la matriz dispersa:

- **Formato coordenado:**

  Refiriéndonos a la implementación con formato coordenado, procedimos a realizar una estructura, la cual posee los tres arreglos respectivos de un tamaño predefinido por nosotros(MAX), en los cuales una lista vacía significa tener el arreglo de **Datos** en su totalidad lleno con ceros y los arreglos de **Columnas** y **Filas**, con números uno negativos esto con el fin de facilitarnos su manipulación, debido que en las posteriores operaciones de la implementación, estos valores negativos en los últimos dos arreglos, servirán a manera de tokens o identificadores de que se ha llegado al final de la lista con números diferentes a cero.

  Sumado a esta facilitación de las funciones longitud de lista y número de elementos a cero, en funciones como lo son suma, se ve una mejoría al momento de trabajarla, al permitirnos simplemente como lo hicimos en nuestro trabajo el meter aquellos números que no tenían una posición o coordenadas iguales y por lo tanto no se realizaba una suma particularmente, sino un modificar elemento en esa posición, agregando aquellos números después de la posición en la que se encuentra el determinante y simplemente organizarlas para dejarla en una representación correcta. También al tener estas características, se hizo más fácil el realizar la función modificar elemento, puesto a que para los casos en que en la matriz el elemento a modificar fuera un cero o se fuera a cambiar por cero, esto solo implicaría el corrimiento de todos los elementos distintos a cero entre la posición en donde gracias a los distintos condicionales se encuentra ahora, hasta la longitud de la lista, ya sea correrlos una posición a la derecha para después insertarlo en una posición normal o correrlos una posición a la izquierda para hacer lo que sería entendido por una modificación a cero.

Particularmente en funciones como sumar y transponer, y como ya se explicó anteriormente, se vio la necesidad de ingresar los datos no sumados inicialmente después del último valor diferente de cero de la lista y pasar a organizarlos, la forma particular de organización que se utilizo fue el algoritmo burbuja por medio de dos funciones, una para simplemente organizar la lista de menor a mayor en medida a sus filas, otra para al ser ejecutada de manera consiguiente a esta se diera un orden de menor a mayor en cuestión a sus columnas; particularmente para la función transponer se utilizó un algoritmo en donde burbuja sería utilizado para dejar los datos en orden ascendente pero en medida de sus columnas, para consecuentemente se intercambien los valores de los arreglos "columnas" y "filas".

- **Formato comprimido:**

  Para esta implementación, al igual que la del *Formato Coordenado,* procedimos creando una estructura que posea los tres arreglos respectivos a la implementación, de un tamaño MAX, donde ahora tanto el arreglo de **Datos** y **Cfilas** estarán llenos en su totalidad con ceros y **Columnas** estará lleno con números uno negativos, este tipo de implementación de la lista se realiza de nuevo, para tener una especie de determinante dentro de la lista en sí de que no existen más números diferentes a cero en esta matriz y por ende también se determina si está vacía u llena absolutamente de ceros.
  Particularmente para esta implementación, gracias a sus características y especificaciones, la acción de modificar elemento se simplificaba en medida de que teníamos un rango más reducido por recorrer de la lista, gracias al arreglo **Cfilas,** a partir de este punto, gracias a los condicionales y el recorrido dado en este rango, se determinaba si era el caso de modificar un dato ya presente la tabla por otro número diferente de cero o de otra manera, pasar a insertarlo o cambiarlo por cero si fuera requerido, para esto se realizó proporcionalmente el proceso realizado en la implementación coordenada, puesto que al tiempo de correr el resto de los arreglos **datos** y **columnas,** ya sea a la izquierda o derecha de igual manera que con los casos de la otra implementación mencionada, **Cfilas** paralelamente cambiara desde la posición de la fila del elemento a modificar más uno hasta la posición que propone la cantidad total de filas más uno, restando o sumando uno a esta posición en **Cfilas** dependiendo si se modifica a 0 un elemento ya existente o si se agrega una posición no existente en la lista respectivamente**.**
  En la función *Transpuesta de una matriz* que implementamos en el *Formato comprimido* el principal problema que se nos presentó es que no teníamos una coordenada para poder hacer la modificación de la posición, pues en esta operación se invierten las filas y las columnas, pero en nuestro caso solo teníamos la información referente a las columnas. Después varios intentos fallidos con la realización de esta función, nos decidimos a analizar bien el comportamiento de las matrices, pues esto nos arrojaría el patrón que nos llevaría a la solución del problema.

Al realizar este análisis, nos dimos cuenta que con esta implementación (Formato comprimido) al realizar la operación transpuesta siempre se pasaban en orden de izquierda a derecha y de menor a mayor (valor de la columna) de la matriz original a la final, para ejemplificar esto está la siguiente tabla:

MATRIZ PRINCIPAL

Valor	1	4	2	8	8	1	1	2	5	6	3	1	2	1	6
Col	1	2	4	0	3	1	4	0	1	2	4	5	0	2	4
Cfilas	0	3	5	7	12	15									

MATRIZ TRANSPUESTA

Valor	8	2	2	1	1	5	4	6	1	8	2	1	3	6	1
Col	1	3	5	0	2	3	0	3	5	1	0	2	3	5	3
Cfilas	0	3	6	9	10	14	15								

Como se puede apreciar en las tablas anteriores, primero se tiene que recorrer toda la lista de columnas preguntando cuánto es el valor de la misma, en el primer ciclo, por ejemplo, si este valor es cero se ingresa a un nuevo arreglo con su respectiva información (Fila, columna y valor), cuando se acaba de recorrer toda la lista, el valor de cero pasa a ser uno y se vuelve a realizar el procedimiento anterior. El índice que cambia a lo largo de los ciclos aumenta hasta que sea igual a la cantidad de columnas que tiene la matriz, por eso esta entra como parámetro a la función.

Finalmente queda realizar la lista de **Cfilas** para poder retornar la matriz final, para esto se realiza un ciclo que recorre el arreglo que acabamos de crear de filas y pregunta si un elemento de esta es igual a un índice que indica la fila en que va, si esto es cierto un contador aumenta en 1, al final de todo el recorrido este valor se suma con el anterior y se agrega en la siguiente posición del arreglo de **Cfilas**. Cuando ya se tiene los nuevos tres arreglos completos, se realiza un último ciclo que pasa esta información a una estructura para poder así retornar la matriz después de haber aplicado la función *Transponer*.

Para la realización de la función sumar en la implementación de *Formato comprimido*, después de diversas pruebas y análisis realizados en el grupo, concluimos que la mejor forma de realizar la suma, era recorriendo los dos arreglos respectivos de cada matriz (arreglos correspondientes a las posiciones de las columnas en base a la cantidad de elementos que hay en cada fila de cada matriz, representados en el arreglo de las filas comprimidas (**Cfilas**) y realizar la suma en base cada uno, empezando por el arreglo de la primera matriz, y culminando en el arreglo de la segunda matriz; al realizar dichos recorridos y haciendo uso de cuatro arreglos diferentes, se almacenan de forma ordenada los valores respectivos de la suma de las matrices y se almacenan junto a ellos sus columnas correspondientes (esto con el fin de no perder ningún dato al recorrer las dos matrices). Al ya poseer los cuatro arreglos con los elementos sumados, hacemos uso de la función *organice*, la cual organiza todos los elementos de los cuatro arreglos en dos arreglos inmersos en un arreglo doble puntero, eliminando todos los elementos iguales; para finalizar, con este nuevo arreglo doble puntero, solamente procedemos a pasar todos esos elementos a una matriz de formato comprimido.

- **Listas Enlazadas por Fila:**

  Con respecto a la implementación de listas enlazadas, podemos decir que se procedió a realizar un arreglo de listas doblemente enlazadas el cual es la base de nuestra implementación, debido a que esto facilita la realización de algunas operaciones, logrando así un mejor producto final. Además, ayuda a reducir el costo de ejecución de algunas operaciones, ejemplo de esto es que la función "Hallar fila" se realice de manera constante.
  De igual manera, la función transponer se facilita en gran medida gracias al tipo de lista usada puesto que se vuelve necesario el realizar un recorrido hasta el final derecho de la lista, para luego hacer el recorrido de manera inversa, pero esta vez ingresándolo en la matriz transpuesta en la fila que sea igual a la columna del elemento por el que vaya el recorrido, el elemento propiamente, esto realizándose en un rango de cero a m, siendo m el número de columnas de la matriz principal.
  La función modificar elemento, se simplifico de manera significativa también al haber tomado la decisión de tomar la estructura de la implementación como un arreglo o vector de listas, puesto a que ahora la operación trataría simplemente de entrar a la posición del arreglo deseado, si es que existe en esta matriz, y recorrer la lista en esta fila hasta que la columna del nodo siguiente al que se apunta, fuera igual a la que se busca modifica, mayor a esta o sea nulo, en el caso de que fuera igual, si se tratara de modificar el valor en este nodo por un cero, se procedería a actualizar la conexiones del nodo siguiente para así técnicamente eliminarlo de la lista, del otro modo, se cambiaría el valor de ese nodo por el propuesto a modificar; en el caso de que sea mayor se procedería a actualizar las conexiones del nodo actual con el siguiente para insertar en medio de estos dos y por consecuencia en la lista el nuevo nodo con el elemento y columna a modificar; en el caso de que el siguiente en la lista fuera NULL, se pasaría a insertarlo en la última posición, es decir, anexar el nodo, cabe

mencionar que la función tiene como precondición que la columna y fila del elemento a modificar pertenezcan al rango propuesto por las columnas y filas de la matriz, por lo tanto no se tiene problema alguno al anexarlo en el último caso mencionado.

Al considerar diversas soluciones para la suma con la implementación de listas enlazadas, consideramos como mejor opción la solución que implementamos. En esta solución consta principalmente de recorrer cada lista de cada fila de forma independiente a la otra mientras que las dos sean diferentes de NULL; al realizar dicha iteración y cumpliendo con algunos condicionales específicos de cada caso posible dentro de la misma, como mencionaba anteriormente, se va recorriendo de forma independiente, comparando así todos los valores correspondientes y con ello agregándolos a la nueva matriz.

2. Análisis de complejidad algorítmica de cada implementación.

Operaciones generales de las implementaciones.	**Implementación coordenada**	**Implementación compresa**	**Implementación con listas**
Crear de Matriz Completa	$O(m+n)$	$O(m*n)$	$O(m*2n)$
Obtener Matriz Completa	$O(n)$	$O(m^2)$	$O(m*2n)$
Obtener Elemento	$O(n)$	$O(n)$	$O(n)$
Obtener Fila	$O(n)$	$O(n)$	$O(1)$
Obtener Columna	$O(n)$	$O(n)$	$O(m*n)$
Obtener Fila Dispersa	$O(n)$	$O(n)$	$O(m+n) \vee O(n^2)$
Obtener Columna Dispersa	$O(n)$	$O(2*m)$	$O(n)$
Obtener número de elementos	$O(n)$	$O(n)$	$O(m)$
Modificar Posición	$O(m(2m+3)+2)$	$O(X-k+ll-n1+m-i-1)$	$O(n)$
Suma de Matrices	$O(5n^2+3n)$	$O(10m^2)$	$O(n^2)$
Producto Matriz Vector	$O(3mn+2m+1)$	$O(m(n+2)+1)$	$O(n(m+1)) \vee O(n^2)$
Matriz Transpuesta	$O(n^2+n+1)$	$O(mn(m^2-m+mn+2))$	$O(n(m+1)) \vee O(n^2)$

Operaciones específicas de la implementación coordenada	**Complejidad**
Crearlista	$O(MAX)$
Crearmatriz1	$O(n*m)$
Imprimir matriz en listas	$O(3n)$
Imprimir matriz de arreglo	$O(m*n)$
Organice	$O(n^2)$
Organice by rows	$O(n^2)$
Organice cols	$O(n^2)$

Operaciones específicas de la implementación con listas	**Complejidad**
Crearlista	$O(1)$
Crearmatriz1	$O(2n*m)$
Imprimir matriz en listas	$O(m*n)$
Imprimir matriz de arreglo	$O(m*n)$
Crearnodo	$O(1)$
CrearMatrizNula	$O(m+1)$
Buscarelelist	$O(n)$
Imprimirlist1	$O(n)$
ImprimirLista	$O(n)$
AnxNodo	$O(n)$

Operaciones específicas de la implementación compresa	**Complejidad**
Crearlista	$O(n)$
Crearmatriz1	$O(2n)$
Imprimir matriz en listas	$O(3n)$
Imprimir matriz de arreglo	$O(m * n)$

3. **conclusiones**

Haciendo la comparación de las complejidades de las diferentes implementaciones, se puede afirmar que la implementación de "Formato comprimido" es la que más recursos requiere para ejecutarse, pues en 5 de las 12 funciones principales fue la que más exigió al computador. Por otro lado, tenemos que la implementación de "Listas enlazadas por fila" fue la más eficiente de todas, estuvo presente en 6 ocasiones dentro de las funciones más eficientes, logrando así quedar en el podio de las implementaciones más eficientes de este proyecto. En base a lo anterior en conjunto a nuestra labor en la realización de dichas funciones podemos concluir que, la implementación de "Formato comprimido" podría considerarse como la implementación que representa mayor dificultad en toda su elaboración; esto se puede deber a las mismas especificaciones de la misma implementación, lo cual dificultó ciertas operaciones como en la forma de recorrer la propia matriz o en modificar algún valor correspondiente a la misma.