UNIDAD V TDA Matriz Dispersa

2.1.1 Descripción del TDA Matriz Dispersa

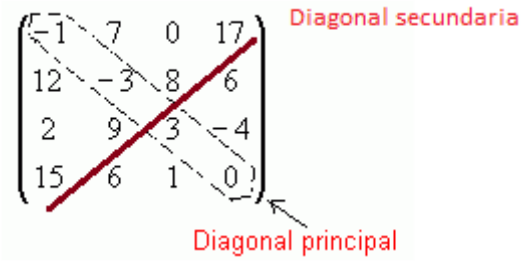
Para describir una matriz dispersa inicialmente definiremos conceptos básicos sobre matrices por lo que una matriz de **orden (m x n)** es un conjunto de $m \times n$ números ordenados en una tabla:

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

en donde podemos apreciar horizontalmente las filas, fila 1: (a_{11} a_{12} $\dots$ a_{1n}), fila 2: (a_{21} a_{22} $\dots$ a_{2n}), etc. Mientras que verticalmente se habla de columnas: columna 1, columna 2, etc.

Por tanto, una matriz de orden (m x n) tiene m filas y n columnas. En caso de que el número de filas y el de columnas sea el mismo se habla de matriz cuadrada

Las matrices cuadradas tienen dos diagonales, de las cuales sobre un ejemplo vemos las que se llamam "diagonal principal" y “diagonal secundaria” de la matriz



Por lo tanto una matriz dispersa es una matriz que en la que la mayoría de sus elementos son cero.

Las matrices dispersas de dimensiones grandes se usan a menudo en ciencia o ingeniería cuando se resuelven ecuaciones diferenciales parciales .
Al implementar matrices dispersas en una computadora , es beneficioso y a menudo necesario utilizar algoritmos especializados y estructuras de datos que aprovechen la estructura dispersa de la matriz. Las definiciones estándar de matrices que se utilizan para programar no se pueden utilizar para matrices dispersas ya que el uso de espacio de memoria seria mayor que la memoria disponible por el computador.

Una matriz dispersa se comporta como una matriz bidimensional, con la característica de que la mayoría de los elementos son cero. Para ahorrar espacio en la memoria, solo se guardan realmente los elementos distintos de cero, junto con suficiente información para guardar o inferir los índices.

Las operaciones básicas con matrices son:

Adición: Sean A y B son dos matrices del mismo orden , entonces la matriz suma S = A + B es:

$$\left.\begin{matrix} A=(a_{ij}) \\ B=(b_{ij}) \end{matrix}\right\} \ (s_{ij})=(a_{ij})+(b_{ij})$$

$$\begin{pmatrix} 1 & 2 & 1 \\ 5 & 3 & 0 \end{pmatrix}+\begin{pmatrix} 3 & -1 & 2 \\ 1 & 0 & -2 \end{pmatrix}=\begin{pmatrix} 4 & 1 & 3 \\ 6 & 3 & -2 \end{pmatrix}$$

Producto por un escalar: Sea A una matriz y k un escalar (un número real), entonces la matriz B = k A es:

$$\left.\begin{matrix} A=(a_{ij}) \\ k\in R \end{matrix}\right\} \ (b_{ij})=(k\cdot a_{ij})$$

$$4\begin{pmatrix} 1 & 2 & 1 \\ 5 & 3 & 0 \end{pmatrix}=\begin{pmatrix} 4 & 8 & 4 \\ 20 & 12 & 0 \end{pmatrix}$$

Considerando esto, podemos hablar de la RESTA de dos matrices A - B, como la suma de A con el producto de (-1)B, lo cual equivale a restar los correspondientes elementos (i,j) de A con los (i,j) de B

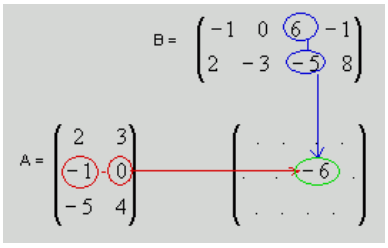
Producto: Sea A una matriz de orden (m x n), y B una matriz de orden (n x r), entonces la matriz producto, es una matriz P = A * B de orden (m ´ r):

$$\left.\begin{matrix} A=(a_{ij}) \\ B=(b_{ij}) \end{matrix}\right\} \ p_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\ldots+a_{in}b_{nj}=\sum_{k=1}^{n}a_{ik}b_{kj}$$

$$A=\begin{pmatrix} 2 & 3 \\ -1 & 0 \\ -5 & 4 \end{pmatrix} \quad , \quad B=\begin{pmatrix} -1 & 0 & 6 & -1 \\ 2 & -3 & -5 & 8 \end{pmatrix} \qquad P=\begin{pmatrix} 4 & -9 & -3 & 22 \\ 1 & 0 & -6 & 1 \\ 13 & -12 & -50 & 37 \end{pmatrix}$$

en la que se han ido obteniendo los elementos multiplicando fila de A por columna de B (por ejemplo):

$$p_{23}=a_{21}b_{13}+a_{22}b_{23}$$
$$-1.6+0.(-5)\rightarrow -6$$



Transpuesta: El concepto de matriz traspuesta no es exclusivo de matrices cuadradas.) Sea una matriz A de orden (m x n), se llama "matriz traspuesta de A", a una matriz, ${}^tA$, de orden (n x m), obtenida a partir de A, cambiando filas por columnas

$$A = (a_{ij}) \quad , \quad {}^TA = (a_{ji}) \qquad A = \begin{pmatrix} 2 & 0 \\ 1 & 5 \\ 3 & 7 \end{pmatrix} \quad , \quad {}^tA = \begin{pmatrix} 2 & 1 & 3 \\ 0 & 5 & 7 \end{pmatrix}$$

Simétrica: Una matriz A es "simétrica" si coincide con su traspuesta, es decir si: $A = {}^tA$. Por ejemplo

$$A = \begin{pmatrix} 1 & 3 & -7 \\ 3 & -2 & 8 \\ -7 & 8 & 0 \end{pmatrix}$$

Identidad: Es una matriz cuadrada (orden n), representada como In, en la que todos sus elementos son 0, excepto los de la diagonal principal, que son unos:

$$I_n = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

2.1.2 Especificación del TDA Matriz Dispersa.

Partiendo de lo establecido en la Unidad 1. Especificación Informal tenemos lo siguiente:
Elementos que conforman la estructura

TDA MatrizDispersa (**VALORES** todos los valores numéricos de cualquier tipo, **OPERACIONES** crear, poner, elemento, dimension_fila, dimension_columna, dimensionar, definir_valor_repetido)

OPERACIONES

crear (M: MatrizDispersa)
Utilidad: Sirve para inicializar la matriz dispersa
Entrada: Matriz Dispersa M
Salida: Ninguna
Precondición: Ninguna.
Poscondición: Matriz Dispersa inicializada 0x0 y valor repetido 0.

Dimensionar(M: MatrizDispersa, df,dc:entero)
Utilidad: Sirve determinar la dimensión de la matriz
Entrada: Matriz Dispersa M
Salida: Ninguna

Precondición: Matriz Dispersa Creada
Poscondición: Matriz Dispersa con dimensión definida (DF x DC.)

Dimension_Fila(M:MatrizDispersa)
Utilidad: Determina cuantas filas máximo maneja la matriz
Entrada: Matriz Dispersa M
Salida: Numero máximo de Filas
Precondición: Matriz Dispersa Creada
Poscondición: Ninguna

Dimension_Columna(M:MatrizDispersa)
Utilidad: Determina cuantas columnas máximo maneja la matriz
Entrada: Matriz Dispersa M
Salida: Número máximo de columnas
Precondición: Matriz Dispersa Creada
Poscondición: Ninguna

Definir_valor_repetido(M:MatrizDispersa, valor: elemento)
Utilidad: Establecer qué valor es el que se repite más en la matriz por defecto "0"
Entrada: Matriz Dispersa M y Valor Repetido
Salida: Ninguna
Precondición: Matriz Dispersa Creada
Poscondición: Matriz Dispersa con valor repetido establecido.

Poner(M:MatrizDispersa, f, c : indice, valor: elemento)
Utilidad: Adicionar un elemento a la matriz
Entrada: Matriz Dispersa M, fila, columna y valor a colocar
Salida: Ninguna
Precondición: Matriz Dispersa Dimensionada
Poscondición: Matriz Dispersa modificada con valor asignado a la posición f,c

Elemento (M:MatrizDispersa, f,c : indice)
Utilidad: Buscar el elemento que está en la posición f,c de la matriz
Entrada: Matriz Dispersa M, fila f y columna c
Salida: Elemento ubicado en la posición f,c de la matriz
Precondición: Matriz Dispersa Dimensionada
Poscondición: Ninguna

2.1.3 Aplicaciones con MatrizDispersa.

En esta sección se puede apreciar que ya estamos en condiciones para plantear algoritmos usando el TDA MatrizDisersa, abstrayéndonos de la forma como esta implementado.

Ej1: Implementar el procedimiento que determine la transpuesta de una matriz A

$$A = (a_{ij}) \quad , \quad {}^{T}A = (a_{ji})$$

$$A = \begin{pmatrix} 2 & 0 \\ 1 & 5 \\ 3 & 7 \end{pmatrix} \quad , \quad {}^{t}A = \begin{pmatrix} 2 & 1 & 3 \\ 0 & 5 & 7 \end{pmatrix}$$

Transpuesta (A :MatrizDispersa , ES tA: MatrizDispersa)
Inicio
tA.dimensionar(a.dimension_columna,a.dimension_fila)
Para cada f=1 hasta a.dimension_fila
Para cada c= hasta a.dimension_columna
tA.poner(c,f,a.elemento(f,c))
fin

2.1.4 Implementaciones del TDA MatrizDispersa

En esta sección mostraremos tres implementaciones para el TDA MatrizDispersa:

- **Implementación con vectores**
- **Implementación con Simulación de Memoria**
- **Implementación con Punteros**

2.1.4.1 Implementación con vectores.

Formato coordenado (COO)

Para la implementación del TDA MatrizDispersa se utilizara tres vectores que contendrán los elementos y los índices respectivamente de la matriz.

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}$$

$$\begin{aligned} \mathrm{vd} &= \begin{bmatrix} 12 & 9 & 7 & 5 & 1 & 2 & 11 & 3 & 6 & 4 & 8 & 10 \end{bmatrix}, \\ \mathrm{vf} &= \begin{bmatrix} 5 & 3 & 3 & 2 & 1 & 1 & 4 & 2 & 3 & 2 & 3 & 4 \end{bmatrix}, \\ \mathrm{vc} &= \begin{bmatrix} 5 & 5 & 3 & 4 & 1 & 4 & 4 & 1 & 1 & 2 & 4 & 3 \end{bmatrix}. \end{aligned}$$

Definiendo la clase MatrizDispersa

Tipo de Datos

Clase MatrizDispersa

Atributos

Vf, // filas
VC, // Columnas
VD : Arreglo(MAX) // elementos
df,dc : Entero // Dimensión
repe : elemento // es el elemento que se repetirá en la matriz
nt : Entero

Metodos

Crear()
dimensionar(df,dc:entero)
entero dimension_Fila()

```
        entero dimension_columna()
        poner(f,c:indice; valor:elemento)
        tipo_elemento Elemento(f,c:indice)
        definir_valor_repetido(valor:elemento)
    fin

Constructor  matrizdispersa.Crear
 inicio
    df=0 dc=0 repe=0 , nt =0
 fin
matrizdispersa.dimensionar(nf, nc : entero)
inicio
  df=nf
  dc=nc
fin

entero matrizdispersa.dimension_fila()
inicio
   retornar df
fin

entero matrizdispersa.dimension_columna()
inicio
   retornar dc
fin

matrizdispersa.poner(f,c: entero; e: Elemento)
Inicio
  Lug = // Buscar en vector vf,vc los valores f y c y retornar indice
  si lug>0 entonces vd[ lug ] = e
       si vd[lug]=rep entonces // desplazar
       caso contrario
         si nt< MAX entoces
                nt = nt +1
                vd[ nt ] = e vf[ nt ] = f vc[ nt ] = c
              caso contrario
                // error no existe espacio
fin
tipo_elemento matrizdispersa.elemento(f,c: entero)
Inicio
   si (f>=1 y f<= df)  y ( c>=1 y c<=dc) entoces
          lug = // buscar f,c en vectores vc,vf y retornar lugar
          si lug>0 entoces
                retornar vd[lug]
              caso contrario
                retornar repe
        caso contrario
          // Error fuera de rango indices
```

Fin

matrizdispersa.Definir_valor_repetido(valor Entero)
inicio
repe=valor
// este algoritmo no considera si este método es llamado en tiempo de ejecución complemente el código.
Fin

Formato Compressed Sparsed Row (CSR)
Para la implementación del TDA MatrizDispersa se utilizará tres vectores que contendrán los elementos y los índices respectivamente de la matriz, a diferencia de lo establecido en el formato COO, esta forma de implementación comprimirá los elementos del vector que tiene los índices de la Fila.

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}$$

$$\begin{aligned} vd &= [\,12\ 9\ 7\ 5\ 1\ 2\ 11\ 3\ 6\ 4\ 8\ 10\,], \\ vf &= [\,5\ 3\ 3\ 2\ 1\ 1\ 4\ 2\ 3\ 2\ 3\ 4\,], \\ vc &= [\,5\ 5\ 3\ 4\ 1\ 4\ 4\ 1\ 1\ 2\ 4\ 3\,]. \end{aligned}$$

Vf(1) = 1
Vf(i +1) – vf(i) = Numero de elementos no repetidos en la fila i

Así, la matriz A en el formato CRS se representa por

$$\begin{aligned} Vd &= [\,1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\,], \\ Vc &= [\,1\ 4\ 1\ 2\ 4\ 1\ 3\ 4\ 5\ 3\ 4\ 5\,], \\ Vf &= [\,1\ 3\ 6\ 10\ 12\ 13\,]. \end{aligned}$$

Definiendo la clase MatrizDispersa
Tipo de Datos
Clase MatrizDispersa
Atributos
VF, // filas
VC, // Columnas
VD : Arreglo(MAX) // elementos
df,dc : Entero // Dimensión
repe : elemento
nt : Entero
Metodos
Privados
Indice Existe_Elemento(f,c:indice) // busca en vd,vc y vf si existe el elemento
Indice Donde_insertar(f,c:indice) // determina donde insertar en vd,vc y vf
publicos

```
        Crear()
        dimensionar(df,dc:entero)
        entero dimension_Fila()
        entero dimension_columna()
        poner(f,c:indice; valor:elemento)
        tipo_elemento Elemento(f,c:indice)
        definir_valor_repetido(valor:elemento)
      fin

Constructor  matrizdispersa.Crear
 inicio
    df=0 dc=0 repe=0 , nt =0
   para cada k=1 hasta (df+1)
         vf[k]=1
    repe=0
    nt=0
  fin
matrizdispersa.dimensionar(nf, nc : entero)
inicio
   df=nf
   dc=nc
fin

entero matrizdispersa.dimension_fila()
inicio
    retornar df
fin

entero matrizdispersa.dimension_columna()
inicio
    retornar dc
fin


Indice matrizdispersa.Existe_Elemento(f,c:indice)
Inicio
      //Verificar si existe elemento
      Existe_lugar=0
      lug_antes=0
      para cada I = 1 to  (f-1)  hacer
              lug_antes=lug_antes + (vf[i+1] - vf[ i ])
      max_elem_fila:=(vf[f+1]-vf[f])
      para cada i=1 to max_elem_fila hacer
          si vc[lug_antes+i]=c entonces
                existe_lugar=lug_antes+i
       Existe_elemento:=existe_lugar;
fin
```

```
Entero matrizdispersa.dondeinsertar(f,c : Indice )
Inicio
      // contando los lugares por fila
       lug_antes=0
       para cada I = 1  hasta (f-1) hacer
              lug_antes=lug_antes + (vf[i+1] - vf[ i ])

      // contando los lugares por columna
       nuevo_lugar=lug_antes
       lugar=lug_antes
       para cada i=1 to (vf[f+1] - vf[ f ]) hacer
        inicio
          lugar=lug_antes+i
          si  C>vc[lugar] entonces nuevo_lugar=lugar
        fin
       nuevo_lugar=nuevo_lugar+1
       Donde_insertar=nuevo_lugar
fin

matrizdispersa.poner(f,c: entero; e: Elemento)
// el siguiente algoritmo no analiza si se coloca el "0"
Inicio
      Lugar =Existe_Elemento(f,c)
      si lugar<>0 entonces
                    inicio
                       vd[lugar]=valor
                    fin
              caso contrario
                inicio
                  lugar=Donde_insertar(f,c)
                  // desplazando vd,vc para insertar nuevo elemento
                  i=nt+1;
                  mientras i>=(lugar+1) hacer
                   inicio
                      vd[i]=vd[i-1]
                      vc[i]=vc[i-1]
                      i=i-1
                   fin
                  vd[lugar]=valor
                  vc[lugar]=c
                  nt=nt+1
                  // ajustando los valores del vector comprimido
                  for i=(f+1) to (df+1) do
                    vf[i]=vf[i]+1
                fin
fin
```

entero matrizdispersa.elemento(f,c: entero)

```
Inicio
    Si (f>=1 y f<=df) y (c>=1 y c<=dc) entonces
     inicio
     lugar=Existe_Elemento(f,c)
     si lugar=0 entoces elem=repe
            caso contrario elem=vd[lugar]
     elemento=elem
     fin
Fin
```

matrizdispersa.Definir_valor_repetido(valor Entero)

```
inicio
   repe=valor
// este algoritmo no considera si este método es llamado en tiempo de ejecución complemente el código.
Fin
```

2.1.4.2 Implementación con Simulación de Memoria (usando la clase CSMemoria)

Esta forma de implementación es netamente académica en virtud a que lo que busca es una mejor comprensión sobre los punteros, para ello se entiende que se usara como Memoria nuestra clase CSmemoria implementada en la unidad uno.

Usando nodos contiguos para cada elemento: en esta forma de implementación los elementos están contenidos en nodos que están almacenados de manera contigua y cuyo rendimiento está en función del orden y el lugar que ocupan los elementos a partir de ptrmatd.

Definiendo la clase MatrizDispersa

Tipo de dato

Nodo
- Fila Entero
- Col Entero
- dato Entero,
- Sig Puntero a Nodo

// fin definición

Dirección Puntero a espacio de memoria de tipo Nodo

Clase Matrizdispersa

Atributos
- PtrMatD Direccion
- rep, dimf,dimc Entero

Metodos
- Crear()
- dimensionar(df,dc:entero)
- entero dimension_Fila()
- entero dimension_columna()
- poner(f,c:indice; valor:elemento)
- tipo_elemento Elemento(f,c:indice)

definir_valor_repetido(valor:elemento)

Fin

Implementación clase MatrizDispersa utilizando Simulador de Memoria CSmemoria.

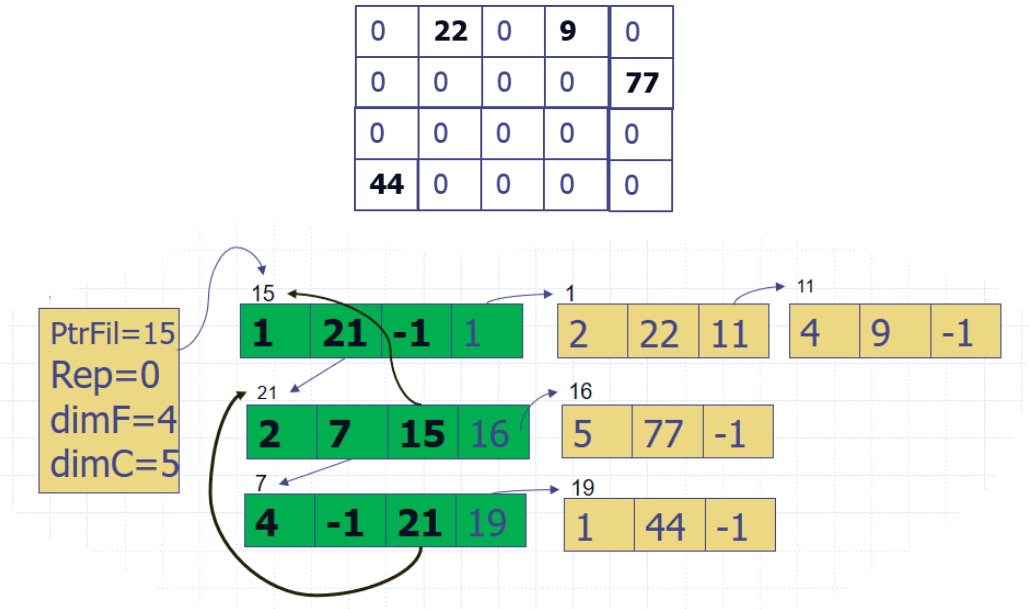
```
publico matrizdispersa.Crear()
  inicio
      ptrmatd=-1
      dimf= 0
      dimc= 0
      rep=0
  fin

matrizdispersa.dimensionar(nf, nc : entero)
inicio
   dimf=nf
   dimc=nc
fin

entero matrizdispersa.dimension_fila()
inicio
    retornar dimf
fin

entero matrizdispersa.dimension_columna()
inicio
    retornar dimc
fin
matrizdispersa.poner(f,c: entero; e: Elemento)
Inicio
        dir= buscar si existe f,c en los nodos
        si dir=nulo entonces
                    x = new_espacio('fila,col,dato,sig')
                    si x<>Nulo entonces
                                poner_dato(x,'->fila',f)
                                poner_dato(x,'->col',c)
                                poner_dato(x,'->dato',e)
                                poner_dato(x,'->sig',ptrmatd)
                                ptrmatd=x
                           caso contrario
                             // error no existe eespacio memoria
                     fin si
               caso contraio
                         poner_dato(dir,'->dato',e)
                         si e=rep entices
                                    //eliminar nodo
                          fin si
          fin si
```

fin

entero matrizdispersa.elemento(f,c: entero)
Inicio
 Si f>=1 y f<=dimf y c>=1 y c<=dimc entoces
  inicio
    dir= buscar si existe f,c en los nodos
    si dir <>nulo entoces
                    retornar obtener_dato(dir,'→dato')
                caso contrario
                  retornar rep
    Fin si
Fin si
Fin

matrizdispersa.Definir_valor_repetido(valor Entero)
inicio
   rep=valor
// este algoritmo no considera si este método es llamado en tiempo de ejecución complemente el código.
Fin

Usando nodos cabecera por fila: En esta forma de implementación los elementos están contenidos en nodos que dependen de un nodo cabera por fila, lo cual hace mas eficiente la búsqueda de los elementos en la matriz.



Definiendo la clase MatrizDispersa
Tipo de dato

NodoF
Fila Entero
SigF Entero // dirección del nodo que tiene la siguiente fila
AntF Entero, // dirección de nodo que tiene la anterior Fila
PtrCol Entero / Direccion que tiene los datos de la columna y elemento
// fin definición
NodoC
Col Entero // Es el valor de la columna donde esta el elemento
Dato Entero // Es el valor de elemento en la matriz
SigCol Entero, // Dirección del próximo elemento para otra columna
// fin definición

Dirección Puntero a espacio de memoria de tipo Nodo
Clase Matrizdispersa
Atributos
PtrFil Direccion
rep, dimf,dimc Entero
Metodos
Crear()
dimensionar(df,dc:entero)
entero dimension_Fila()
entero dimension_columna()
poner(f,c:indice; valor:elemento)
tipo_elemento Elemento(f,c:indice)
definir_valor_repetido(valor:elemento)
Fin

Implemente los métodos de la clase matrizdispersa, considerando que están implementada con nodos cabeceras por filas según la gráfica planteada.

2.1.4.3 Implementación con punteros.
En esta forma de implementación planteada lo que se resalta son los cambios que tienen que hacerse al código de la implementación con el simulador de memoria considerando que ahora se está trabajando con punteros reales, es así que se tiene de color rojo los cambios fundamentales en los algoritmos ya vistos, quedando por resolver las definiciones formales en c++.
Definiendo la clase MatrizDispersa
Tipo de dato

Nodo
Fila Entero
Col Entero
dato Entero,
Sig Puntero a Nodo
// fin definición

Dirección Puntero a espacio de memoria de tipo Nodo

Clase Matrizdispersa

Atributos

PtrMatD Direccion a tipo Nodo

rep, dimf,dimc Entero

Metodos

Crear()

dimensionar(df,dc:entero)

entero dimension_Fila()

entero dimension_columna()

poner(f,c:indice; valor:elemento)

tipo_elemento Elemento(f,c:indice)

definir_valor_repetido(valor:elemento)

Fin

Implementación clase MatrizDispersa utilizando Simulador de Memoria CSmemoria.

```
publico matrizdispersa.Crear()
  inicio
      ptrmatd=null
      dimf= 0
      dimc= 0
      rep=0
  fin

matrizdispersa.dimensionar(nf, nc : entero)
inicio
   dimf=nf
   dimc=nc
fin

entero matrizdispersa.dimension_fila()
inicio
    retornar dimf
fin

entero matrizdispersa.dimension_columna()
inicio
    retornar dimc
fin
matrizdispersa.poner(f,c: entero; e: Elemento)
Inicio
        dir= buscar si existe f,c en los nodos
        si dir=nulo entonces
                    x = new Nodo
                    si x<>Nulo entonces
                                x->fila= f
```

```
                                x->col=c
                                x->dato=e
                                x->sig=ptrmatd
                                ptrmatd=x
                                nt=nt +1
                            caso contrario
                              // error no existe eespacio memoria
                    fin si
                caso contraio
                        dir->dato=e
                        si e=rep entices
                                //eliminar nodo
                                 nt = nt -1
                         fin si
        fin si
fin
```

entero matrizdispersa.elemento(f,c: entero)

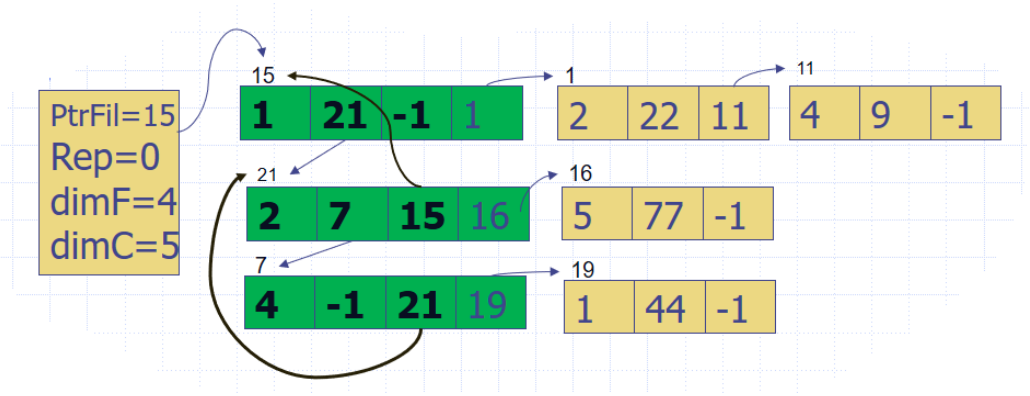
```
Inicio
 Si f>=1 y f<=dimf y c>=1 y c<=dimc entoces
  inicio
    dir= buscar si existe f,c en los nodos
    si dir <>nulo entoces
                    retornar dir→dato
                caso contrario
                  retornar rep
    Fin si
Fin si
Fin
```

matrizdispersa.Definir_valor_repetido(valor Entero)

```
inicio
   rep=valor
 // este algoritmo no considera si este método es llamado en tiempo de ejecución
complemente el código.
Fin
```

Usando nodos cabecera por fila: En esta forma de implementación los elementos están contenidos en nodos que dependen de un nodo cabera por fila, lo cual hace mas eficiente la búsqueda de los elementos en la matriz.

0	**22**	0	**9**	0
0	0	0	0	**77**
0	0	0	0	0
44	0	0	0	0



Definiendo la clase MatrizDispersa
Tipo de dato

NodoF
Fila Entero // Indica la fila de la matriz
SigF Puntero a NodoF // indica la dirección del nodo que tiene la siguiente fila
AntF Puntero a NodoF // inidica la dirección de nodo que tiene la anterior Fila
PtrCol Puntero a NodoC //que contiene los datos de la columna y elemento
// fin definición
NodoC
Col Entero // Es el valor de la columna donde esta el elemento
Dato Entero // Es el valor de elemento en la matriz
SigCol Puntero a nodoC, // Dirección del próximo elemento para otra columna
// fin definición

Dirección Puntero a espacio de memoria de tipo Nodo
Clase Matrizdispersa
Atributos
PtrFil Direccion
rep, dimf,dimc Entero
Metodos
Crear()
dimensionar(df,dc:entero)
entero dimension_Fila()
entero dimension_columna()
poner(f,c:indice; valor:elemento)
tipo_elemento Elemento(f,c:indice)
definir_valor_repetido(valor:elemento)
Fin

Implemente los métodos de la clase matrizdispersa, considerando que están implementada con nodos cabeceras por filas según la gráfica planteada.

Practica
Implementar el TDA Matriz dispersa usando vectores, formato COO
Implementar el TDA Matriz dispersa usando vectores, formato CSR
Implementar el TDA Matriz dispersa usando simulador Memoria, formato Nodos Contiguos

Implementar el TDA Matriz dispersa usando punteros, formato Nodos Contiguos
Implementar el TDA Matriz dispersa usando SMemoria, formato nodos cabecera por fila
Implementar el TDA Matriz dispersa usando Punteros, formato nodos cabecera por fila