Proyecto Final – Análisis y Diseño de Algoritmos.

Juan Diego Escobar Triviño-2359393
Julio David Cardona Melendez-2359654

Mateo Echeverry Correa, Ing.

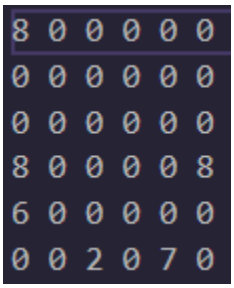
2024
Universidad del valle
Sede Tulua

Representaciones de Matrices Dispersas

1. COO:

Formato:

Este formato utiliza tres espacios para almacenar únicamente los valores diferentes de cero, junto a sus coordenadas, en este caso fila y columna.
Para el ejemplo de prueba utilizaremos la siguiente matriz:



El formato COO, sería:

- Valores:[8,8,8,6,2,7]
- Filas:[0,3,3,4,5,5]
- Columnas:[0,0,5,0,2,4]

Esto desde la parte teórica

Proceso desde Código (Python):

Tenemos una función llamada Representacion_coo que recibe en un archivo txt, en el cual se encuentra la matriz a representar:

```
def Representacion_coo(archivo):
    vals = []
    fil = []
    cols = []
    try:
        with open(f"TxtPruebas/{archivo}", "r") as f:
```

Lo primero que hace es leer el archivo línea por línea, para ello, cada línea se convierte la fila en una lista de número enteros:

```
for i, linea in enumerate(f):
    fila = map(int, linea.split())
```

(Este punto se repite en todos los tres casos así que se tomará como ya hecho en la explicación para no caer en redundancia y lo llamaremos paso uno).

Se itera a través de la listas y si un valor es diferente de cero, se agrega a las listas, el valor a vals(valores), el índice de la fila a fil(filas), el índice de la columna a cols(columnas):

```
for j, val in enumerate(fila):
    if val != 0:
        vals.append(val)
        fil.append(i)
        cols.append(j)
```

Después se devuelve un diccionario con los arreglos o vectores de esta forma:

```
return {"valores": vals, "filas": fil, "columnas": cols}
```

Complejidad:

En cuanto a la complejidad del Algoritmo, es O(n) ya que esto se debe a que se recorre cada elemento de la matriz una vez y se procesan solo los valores distintos de cero en tiempo constante

2. CSR

Formato:

Este formato al igual que el anterior almacena los datos en tres espacios pero, la diferencia se encuentra en que en vez de vector filas, tiene el vector p-filas que almacenará la posición donde inician los valores de cada fila en el segundo vector (columnas).Para el ejemplo de prueba utilizaremos la siguiente

```
7 0 0 0 10 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
6 0 0 0 0 0
0 0 0 0 0 1
```

El formato CSR seria:

- Valores:[7,10,6,1]
- Columnas:[0,4,0,5]
- p-filas[0,2,2,2,2,3,4]

Esto desde la parte teórica

Proceso desde Código (Python):

Para la representación de la matriz en CSR tenemos una función llamada Representacion_csr, dentro de ella contiene el paso uno del anterior punto, pero con una modificación para que cuando encuentre una fila totalmente vacía, pare de iterar linea por linea:

```
linea_compl = linea.lstrip()
if not linea_compl or not linea_compl[0].isdigit():
    continue
```

Para implementar el CSR se utilizó como base el código para el anterior pero añadiendo contador para que por cada iteración de línea lo agregara a la lista o vector p-filas, para actualizar el valor del contador se agregar un cont += 1 dentro del bloque que verifica los valores diferentes de cero,

esto para que cada que se agregue un número a las lista se tome los valores correspondientes.:

```
for i, linea in enumerate(f):
    #Verifica que no este vacia la linea, me mate horas
    linea_compl = linea.lstrip()
    if not linea_compl or not linea_compl[0].isdigit():
        continue

    fila = map(int, linea.split())
    p_fil.append(cont)
    for j, val in enumerate(fila):
        if val != 0:
            vals.append(val)
            cols.append(j)
            cont += 1
p_fil.append(cont)
```

Al final de las iteraciones se agrega por última vez el contador para así devolver el diccionario con los arreglos:

```
return {"valores": vals, "columnas": cols,"filas": p_fil}
```

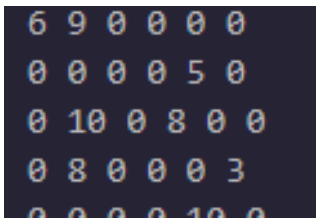
Complejidad:

En cuanto a la complejidad del Algoritmo, es O(n), donde n es el número total de elementos de la matriz. Esto se debe a que recorre toda la matriz una sola vez para identificar los valores no nulos

3. CSC

Formato

Este formato al igual que el anterior almacena los datos en tres espacios pero, la diferencia se encuentra en que en vez de vector columnas, tiene el vector p-columna que almacenará la posición donde inician los valores



de cada columna en el segundo vector (filas).Para el ejemplo de prueba utilizaremos la siguiente matriz:

El formato CSC seria:

- Valores:[6,9,10,8,8,5,10,3,8]
- Filas:[0,0,2,3,2,1,4,3,5]
- p-columnas[0,1,4,4,5,7,9]

Esto desde la parte teórica

Proceso desde Código (Python):

Tenemos una función llamada Representacion_csc que recibe un archivo donde se encuentra la matriz que queremos representar.

```
def Representacion_csc(archivo):
    vals = []
    p_col = [0]
    cols = []
    cont = 0
    try:
        matr = []
        with open(f"TxtPruebas/{archivo}", "r") as f:
```

A diferencia de los anteriores ahora se debe leer la lista de por filas y no por columnas, para ello lo que hacemos es volver armar la matriz, para más facilidad a la hora de leer por filas. Al igual que en el anterior si encuentra una línea vacía para iterar a través del txt.

Después de armar la matriz, obtenemos el número de filas y columnas, para luego iterar sobre la columnas y filas con dos iteradores, anidados,

el iterador interno será el de las filas que verifica si el valor actual en la matriz es diferente de cero para añadirlo la las lista y cont += 1(cont para llevar la cuenta para p-columnas), después de iterar sobre las filas se añade el cont a la lista cont. Ya por último se devuelve el diccionario con las lista.

```
        for linea in f:
            lin_compl = linea.lstrip()
            if not lin_compl or not lin_compl[0].isdigit():
                continue
            fila = list(map(int, linea.split()))
            matr.append(fila)
    num_fil = len(matr)
    num_col = len(matr[0]) if matr else 0
    for j in range(num_col):
        for i in range(num_fil):
            val = matr[i][j]
            if val != 0:
                vals.append(val)
                cols.append(i)
                cont += 1
        p_col.append(cont)
    return {"val": vals, "fil": cols,"p_col": p_col}
```

Complejidad:

En cuanto a la complejidad del Algoritmo, es O(n · m), ya que recorre toda la matriz por columnas.

Descripción de las operaciones implementadas

1. En COO
    a. Crear una matriz a partir de una representación:

```
def Apartir_COO(repre):
    max_fil = max(repre["filas"])+1
    max_col = max(repre["columnas"]) + 1
    matr = [[0 for _ in range(max_col)] for _ in range(max_fil)]
    for valor, i, j in zip(repre["valores"], repre["filas"], repre["columnas"]):
        matr[i][j] = valor
    return matr
```

Tenemos una función llamada Apartir_COO() que recibe la representación de una matriz en formato COO, de la representación se obtienen las filas y las columnas, luego se crea una matriz del tamaño

necesario llena de ceros, para finalmente a través de un for rellenarlo validando pro tuplas su valor y posición.

b. Obtener un elemento dados sus índices i,j

```
def Obte(repre, i, j):
    for val, fila, col in zip(repre["valores"], repre["filas"], repre["columnas"]):
        if fila == i and col == j:
            return val
    return 0
```

Tenemos una función llamada Obte() que recibe la representación de la matriz en COO, las coordenadas i(fila), y j(columna), a través de un for verificando por tuplas el valor, la fila y la columna, busca el valor en la representación con if con las condiciones que se dieron, para finalmente devolver el valor del elemento encontrado o cero si no lo encuentra.

c. Obtener una fila dado su índice i.

```
def Obtfila(repre, i):
    max_col = max(repre["columnas"]) + 1
    fil = [0] * max_col
    for val, fil_act, col in zip(repre["valores"], repre["filas"], repre["columnas"]):
        if fil_act == i:
            print(val)
            print(fil_act)
            print(col)
            fil[col] = val
    return fil
```

Tenemos una función llamada Obtfila() que recibe la representación de la matriz en formato COO, así como la coordenada de la fila que se desea obtener. Luego, se obtiene el número de columnas para crear un array lleno de ceros con el mismo número de columnas. Después, mediante un bucle `for` que recorre las tuplas, verificamos el valor, la fila y la columna. Dentro del bucle, se verifica si estamos en la fila deseada para completar el array con los valores de esa fila. Finalmente, se retorna esa fila (fil)

d. Obtener una columna dado su índice j

```python
def ObtCol(repre, j):
    max_fila = max(repre["filas"]) + 1
    col = [0] * max_fila
    for valor, fila, col_actual in zip(repre["valores"], repre["filas"], repre["columnas"]):
        if col_actual == j:
            col[fila] = valor
    return col
```

Tenemos una función llamada ObtCol() que recibe la representación de la matriz en COO, la coordenada de la columna a obtener, luego se obtiene el número de filas, para crear un array lleno de ceros con las mismas filas, después con un for a traves de tuplas verificamos, el valor, la fila y columna, dentro del for se verifica si estamos en la columna deseada, para completar el array que creamos con los valores de la columna, para después retornar la columna(col).

e. Modificar una posición dados sus índices i,j y un valor x.

```python
def Modele(repre, i, j, x):
    encontrado = False
    for idx, (fil, col) in enumerate(zip(repre["filas"], repre["columnas"])):
        if fil == i and col == j:
            encontrado = True
            if x == 0:
                del repre["valores"][idx]
                del repre["filas"][idx]
                del repre["columnas"][idx]
            else:
                repre["valores"][idx] = x
            break
    if not encontrado and x != 0:
        repre["valores"].append(x)
        repre["filas"].append(i)
        repre["columnas"].append(j)
```

Tenemos una función llamada Modele() que recibe la representación de la matriz en COO, las coordenadas i(fila) y j(columna), y el valor x que queremos asignar en esa posición. Primero se inicializa una variable encontrado en falso, luego con un for a través de tuplas verificamos la fila y la columna junto con su índice, dentro del for se verifica si estamos en la posición deseada, si se encuentra y x es igual a 0 se eliminan el valor, fila y columna de las listas correspondientes, si x no es 0 se actualiza el

valor en la posición, si no se encuentra y x es diferente de 0 se agrega el nuevo valor junto con su fila y columna al final de las listas.

f. Suma de matrices

g. Matriz Transpuesta

2. En CSR

a. Crear una matriz a partir de una representación:

```
def Apartir_csr(repre):
    max_fil = len(repre["filas"])-1
    max_col = max(repre["columnas"]) + 1
    matr = [[0 for _ in range(max_col)] for _ in range(max_fil)]
    for i in range(max_fil):
        ini = repre["filas"][i]
        fin = repre["filas"][i +1]
        for j in range(ini, fin):
            col = repre["columnas"][j]
            matr[i][col] = repre["valores"][j]
    return matr
```

La función Apartir_csr() se encarga de crear una matriz a partir de una representación en formato CSR, primero determina el número máximo de filas y columnas a partir de los datos proporcionados en la representación, luego, crea una matriz llena de ceros con las dimensiones adecuada, después, utiliza un bucle para recorrer cada fila, identificando el inicio y el fin de los elementos no nulos en la representación, para cada elemento, extrae la columna y asigna su valor en la posición adecuada de la matriz, finalmente, devuelve la matriz completa.

b. Obtener un elemento dados sus índices i,j

```
def Obte(repre, i, j):
    ini = repre["filas"][i]
    f = repre["filas"][i+1]
    for tj in range(ini, f):
        print(tj)
        if repre["columnas"][tj] == j:
            return repre["valores"][tj]
    return 0
```

La función Obte() se utiliza para obtener el valor de un elemento en una matriz representada en formato CSR, recibe como parámetros la representación de la matriz, así como las coordenadas de fila i y columna j.Primero, determina el índice inicial y final de los elementos no nulos en la fila i utilizando el arreglo de fila, después itera los índices de los elementos en esa fila,durante la iteración, verifica si la columna coincide con j, si encuentra una coincidencia, devuelve el valor de esa posición, si no se encuentra el valor, la función retorna 0, indicando que el elemento es cero en la matriz original.

c. Obtener una fila dado su índice i.

```
def Obtfila(repre, i):
    max_col = max(repre["columnas"]) + 1
    fil = [0] * max_col
    ini = repre["filas"][i]
    f = repre["filas"][i+1]
    for k in range(ini, f):
        col = repre["columnas"][k]
        fil[col] = repre["valores"][k]
    return fil
```

La función Obtfila() obtiene una fila completa de una matriz representada en formato CSR, recibe como parámetros la representación de la matriz y el índice de la fila que se desea extraer, primero, calcula el número máximo de columnas a partir del arreglo de columnas y crea una lista llena de ceros para almacenar los valores de la fila, luego, determina los índices de inicio y fin de los elementos no nulos en la fila utilizando el arreglo de filas. Después, itera sobre estos índices, extrayendo las columnas correspondientes y asignando los valores a su posición en la lista ya de ultimo, devuelve la lista que representa la fila completa, con ceros en las posiciones donde no hay elementos no nulos.

d. Obtener una columna dado su índice j

```python
def ObtCol(repre, j):
    max_fila = len(repre["filas"]) - 1
    col = [0] * max_fila

    for fila in range(max_fila):
        inicio = repre["filas"][fila]
        fin = repre["filas"][fila + 1]

        for k in range(inicio, fin):
            if repre["columnas"][k] == j:
                col[fila] = repre["valores"][k]
                break
    return col
```

La función ObtCol() se encarga de extraer una columna completa de una matriz representada en formato CSR, recibe como parámetros la representación de la matriz y el índice de la columna que se desea obtener, determina el número máximo de filas a partir del arreglo de filas y crea una lista en ceros para almacenar los valores de la columna,para despues, iterar sobre cada fila, identificando los índices de inicio y fin de los elementos no nulos en esa fila, dentro este rango, busca si la columna correspondiente coincide con j. Si encuentra una coincidencia, asigna el valor correspondiente a la posición adecuada en la lista de la columna y utiliza break para salir del bucle, ya que solo necesita el primer valor encontrado. Finalmente, devuelve la lista que representa la columna completa, con ceros en las posiciones donde no hay elementos no nulos.

e. Modificar una posición dados sus índices i,j y un valor x

```python
def Modele(repre, i, j, x):
    encontrado = False
    for idx in range(repre["filas"][i], repre["filas"][i + 1]):
        if repre["columnas"][idx] == j:
            encontrado = True
            if x == 0:
                del repre["valores"][idx]
                del repre["columnas"][idx]
                for k in range(i + 1, len(repre["filas"])):
                    repre["filas"][k] -= 1
            else:
                repre["valores"][idx] = x
            break
    if not encontrado and x != 0:
        repre["valores"].append(x)
        repre["columnas"].append(j)
        for k in range(i + 1, len(repre["filas"])):
            repre["filas"][k] += 1
        repre["filas"].append(repre["filas"][-1])
```

La función Modele() se encarga de modificar un elemento en una matriz representada en formato CSR. Recibe como parámetros la representación de la matriz, las coordenadas de fila y columna y el nuevo valor x que se desea establecer, busca si ya existe un elemento en la posición especificada, si lo encuentra y el valor x es cero, elimina el valor y la columna correspondientes, ajustando el arreglo de filas para reflejar la eliminación, si el valor x no es cero, actualiza el valor existente en esa posición, si no encuentra el elemento y x es diferente de cero, agrega el nuevo valor y su columna al final de los arreglos de valores y columnas, respectivamente, y ajusta el arreglo de filas para incluir este nuevo elemento, además, se asegura de que las filas posteriores se actualicen correctamente para mantener la integridad de la representación CSR.

f. Suma de matrices
g. Matriz Transpuesta

3. En CSC
    a. Crear una matriz a partir de una representación:

```python
def Apartir_csc(repre):
    max_col = len(repre["p_col"]) - 1
    print(max_col)
    max_fil = max(repre["fil"]) + 1
    matr = [[0 for _ in range(max_col)] for _ in range(max_fil)]

    for col in range(max_col):
        ini = repre["p_col"][col]
        fin = repre["p_col"][col + 1]
        for j in range(ini, fin):

            fila = repre["fil"][j]
            matr[fila][col] = repre["val"][j]
    return matr
```

La función Apartir_csc() se utiliza para crear una matriz a partir de una representación en formato CSC, recibe como parámetro la representación de la matriz, que incluye punteros a las columnas, filas y valores, primero, determina el número máximo de columnas a partir del arreglo de punteros de columna y calcula el número máximo de filas utilizando el valor máximo del arreglo de filas más uno, se crea una matriz llena de ceros con las dimensiones adecuadas.

Después, itera sobre cada columna, identificando los índices de inicio y fin de los elementos no nulos en esa columna utilizando el arreglo de punteros. Durante esta iteración, extrae la fila correspondiente y asigna el valor a la posición adecuada en la matriz. Finalmente, devuelve la matriz completa que representa la estructura original en formato denso.

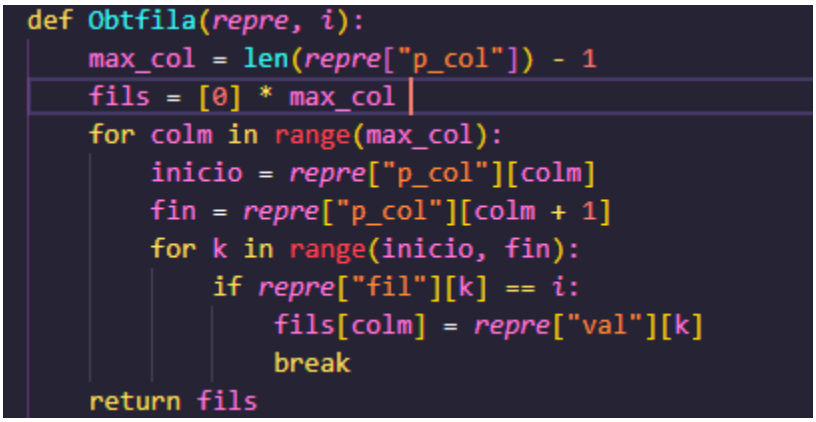
b. Obtener un elemento dados sus índices i,j

```python
def Obte(repre, i, j):
    ini = repre["p_col"][i]
    f = repre["p_col"][i+1]
    for tj in range(ini, f):
        print(tj)
        if repre["fil"][tj] == j:
            return repre["val"][tj]
    return 0
```

La función Obte() tiene como objetivo obtener el valor de un elemento en una matriz representada en formato CSC. Recibe como parámetros la

representación de la matriz, así como las coordenadas de fila j y columna i. Primero, determina los índices de inicio y fin de los elementos no nulos en la columna i utilizando el arreglo de punteros a columnas. Luego, itera sobre estos índices, buscando si la fila correspondiente coincide con j. Si encuentra una coincidencia, devuelve el valor asociado a esa posición. Si no se encuentra el valor, la función retorna 0, indicando que el elemento es cero en la matriz original.

c. Obtener una fila dado su índice i.



```
def Obtfila(repre, i):
    max_col = len(repre["p_col"]) - 1
    fils = [0] * max_col
    for colm in range(max_col):
        inicio = repre["p_col"][colm]
        fin = repre["p_col"][colm + 1]
        for k in range(inicio, fin):
            if repre["fil"][k] == i:
                fils[colm] = repre["val"][k]
                break
    return fils
```

La función Obtfila() se encarga de obtener una fila específica de una matriz representada en formato CSC. Toma como parámetros la representación de la matriz y el índice de la fila i que queremos extraer. Primero, se determina el número máximo de columnas a partir del arreglo de punteros a columnas y se inicializa una lista llamada fils con ceros para almacenar los valores de la fila. Luego, se itera sobre cada columna, obteniendo los índices de inicio y fin de los elementos no nulos en esa columna. Dentro del bucle, se verifica si la fila correspondiente es igual a . Si es así, se asigna el valor correspondiente a la posición adecuada en la lista fils y se rompe el bucle para continuar con la siguiente columna. Al final, se devuelve la lista fils que representa la fila completa, con ceros donde no hay elementos no nulos.

d. Obtener una columna dado su índice j

```
def ObtCol(repre, j):
    max_fil = max(repre["fil"]) + 1
    col = [0] * max_fil
    ini = repre["p_col"][j]
    f = repre["p_col"][j+1]
    for k in range(ini, f):
        fil = repre["fil"][k]
        col[fil] = repre["val"][k]
    return col
```

La función ObtCol() se encarga de retornar una columna específica, de una matriz representada en formato CSC , recibe como parámetros la representación de la matriz y el índice de la columna j que queremos extraer. Primero, se determina el número máximo de filas a partir del arreglo de filas y se inicializa una lista llamada col con ceros para almacenar los valores de la columna. Luego, se obtienen los índices de inicio y fin de los elementos no nulos en la columna j utilizando el arreglo de punteros a columnas. A continuación, se itera sobre estos índices, extrayendo la fila correspondiente y asignando el valor a la posición adecuada en la lista col. Finalmente, se devuelve la lista col que representa la columna completa, con ceros en las posiciones donde no hay elementos no nulos.

e. Modificar una posición dados sus índices i,j y un valor x

```python
def Modele(repre, i, j, x):
    encontrado = False

    for idx in range(repre["p_col"][i], repre["p_col"][i + 1]):
        if repre["fil"][idx] == j:
            encontrado = True
            if x == 0:
                del repre["val"][idx]
                del repre["fil"][idx]
                for k in range(i + 1, len(repre["p_col"])):
                    repre["p_col"][k] -= 1
            else:
                repre["val"][idx] = x
            break
    if not encontrado and x != 0:
        repre["val"].append(x)
        repre["fil"].append(j)
        for k in range(i + 1, len(repre["p_col"])):
            repre["p_col"][k] += 1
        repre["p_col"].append(repre["p_col"][-1])
```

La función Modele() se encarga de modificar un elemento en una matriz representada en formato COO, recibe como parámetros la representación de la matriz, las coordenadas i(fila) y j(columna), y el nuevo valor x que se desea establecer. Primero, inicializa una variable encontrada en False para rastrear si el elemento ya existe. Luego, itera sobre las tuplas de filas y columnas a través de los arreglos de punteros. Si encuentra que la fila correspondiente es igual a i y la columna es igual a j, establece encontrado en True. Si el nuevo valor x es igual a 0, elimina el valor, la fila y la columna correspondientes de los arreglos, ajustando los punteros para reflejar la eliminación. Si x no es 0, simplemente actualiza el valor en esa posición. Si no encuentra el elemento y x es diferente de 0, agrega el nuevo valor, la fila y la columna al final de los arreglos de valores, filas y columnas, respectivamente, manteniendo la integridad de la representación COO.

f. Suma de matrices

La suma de matrices se realiza por separado para cada uno de los formatos, dado que se trabajara basándonos en cada uno de los formatos de salida de manera individual.

Suma formato Coo:

```python
def SumarMatricesCOO(matriz1, matriz2):
    resultado = {
        "valores": [],
        "filas": [],
        "columnas": []
    }

    # Agregar todos los elementos de la primera matriz al resultado
    for val, fila, col in zip(matriz1["valores"], matriz1["filas"], matriz1["columnas"]):
        resultado["valores"].append(val)
        resultado["filas"].append(fila)
        resultado["columnas"].append(col)

    # Agregar los elementos de la segunda matriz al resultado
    for val, fila, col in zip(matriz2["valores"], matriz2["filas"], matriz2["columnas"]):
        encontrado = False
        for idx, (res_fila, res_col) in enumerate(zip(resultado["filas"], resultado["columnas"])):
            if res_fila == fila and res_col == col:
                resultado["valores"][idx] += val
                encontrado = True
                break
        if not encontrado:
            resultado["valores"].append(val)
            resultado["filas"].append(fila)
            resultado["columnas"].append(col)

    return resultado
```

Inicialización del resultado: Se crea un diccionario resultado con listas vacías para valores, filas y columnas.

Agregar elementos de la primera matriz: Se recorren los elementos de la primera matriz (matriz1) y se agregan directamente al resultado.

Agregar elementos de la segunda matriz: Se recorren los elementos de la segunda matriz (matriz2). Para cada elemento, se verifica si ya existe en la misma posición en el resultado:

Si existe, se suman los valores.
Si no existe, se agrega el nuevo elemento al resultado.
Devolver el resultado: Se devuelve el diccionario resultado que contiene la suma de las dos matrices en formato COO.

Formato CSC:

```python
def SumarMatricesCSC(matriz1, matriz2):
    resultado = {
        "val": [],
        "fil": [],
        "p_col": [0]
    }

    num_cols = max(len(matriz1["p_col"]), len(matriz2["p_col"])) - 1
    cont = 0

    for j in range(num_cols):
        col1_start = matriz1["p_col"][j] if j < len(matriz1["p_col"]) - 1 else 0
        col1_end = matriz1["p_col"][j + 1] if j + 1 < len(matriz1["p_col"]) else 0
        col2_start = matriz2["p_col"][j] if j < len(matriz2["p_col"]) - 1 else 0
        col2_end = matriz2["p_col"][j + 1] if j + 1 < len(matriz2["p_col"]) else 0

        col1_vals = {matriz1["fil"][i]: matriz1["val"][i] for i in range(col1_start, col1_end)}
        col2_vals = {matriz2["fil"][i]: matriz2["val"][i] for i in range(col2_start, col2_end)}

        all_rows = set(col1_vals.keys()).union(set(col2_vals.keys()))

        for row in sorted(all_rows):
            val = col1_vals.get(row, 0) + col2_vals.get(row, 0)
            if val != 0:
                resultado["val"].append(val)
                resultado["fil"].append(row)
                cont += 1

        resultado["p_col"].append(cont)

    return resultado
```

La función SumarMatricesCSC suma dos matrices dispersas en formato CSC (Compressed Sparse Column). Aquí tienes una breve descripción de cómo funciona:

Inicialización del resultado: Se crea un diccionario resultado con listas vacías para val, fil, y p_col. La lista p_col se inicializa con un solo elemento 0.

Determinar el número de columnas: Se calcula el número máximo de columnas (num_cols) entre las dos matrices de entrada.

Recorrer cada columna:

Para cada columna j, se obtienen los índices de inicio y fin de los elementos no nulos en esa columna para ambas matrices (matriz1 y matriz2).
Se crean diccionarios (col1_vals y col2_vals) que mapean las filas a sus valores correspondientes en la columna j para ambas matrices.
Sumar los valores de las columnas:

Se obtiene el conjunto de todas las filas (all_rows) que tienen valores no nulos en la columna j de ambas matrices.
Para cada fila en all_rows, se suman los valores correspondientes de col1_vals y col2_vals. Si la suma no es cero, se agrega el valor resultante a la lista val del resultado, y la fila correspondiente a la lista fil.
Actualizar los punteros de columna:

Después de procesar cada columna, se actualiza el puntero de columna p_col en el resultado con el contador cont, que lleva la cuenta del número total de elementos no nulos procesados hasta el momento.
Devolver el resultado: Se devuelve el diccionario resultado que contiene la suma de las dos matrices en formato CSC.

Formato CSR:

```python
def SumarMatricesCSR(matriz1, matriz2):
    resultado = {
        "valores": [],
        "columnas": [],
        "filas": [0]
    }

    num_filas = max(len(matriz1["filas"]), len(matriz2["filas"])) - 1
    cont = 0

    for i in range(num_filas):
        fila1_start = matriz1["filas"][i] if i < len(matriz1["filas"]) - 1 else 0
        fila1_end = matriz1["filas"][i + 1] if i + 1 < len(matriz1["filas"]) else 0
        fila2_start = matriz2["filas"][i] if i < len(matriz2["filas"]) - 1 else 0
        fila2_end = matriz2["filas"][i + 1] if i + 1 < len(matriz2["filas"]) else 0

        fila1_vals = {matriz1["columnas"][j]: matriz1["valores"][j] for j in range(fila1_start, fila1_end)}
        fila2_vals = {matriz2["columnas"][j]: matriz2["valores"][j] for j in range(fila2_start, fila2_end)}

        all_cols = set(fila1_vals.keys()).union(set(fila2_vals.keys()))

        for col in sorted(all_cols):
            val = fila1_vals.get(col, 0) + fila2_vals.get(col, 0)
            if val != 0:
                resultado["valores"].append(val)
                resultado["columnas"].append(col)
                cont += 1

        resultado["filas"].append(cont)

    return resultado
```

Inicialización del resultado: Se crea un diccionario resultado con listas vacías para valores, columnas, y filas. La lista filas se inicializa con un solo elemento 0.

Determinar el número de filas: Se calcula el número máximo de filas (num_filas) entre las dos matrices de entrada.

Recorrer cada fila:

Para cada fila i, se obtienen los índices de inicio y fin de los elementos no nulos en esa fila para ambas matrices (matriz1 y matriz2).
Se crean diccionarios (fila1_vals y fila2_vals) que mapean las columnas a sus valores correspondientes en la fila i para ambas matrices.
Sumar los valores de las filas:

Se obtiene el conjunto de todas las columnas (all_cols) que tienen valores no nulos en la fila i de ambas matrices.

Para cada columna en all_cols, se suman los valores correspondientes de fila1_vals y fila2_vals. Si la suma no es cero, se agrega el valor resultante a la lista valores del resultado, y la columna correspondiente a la lista columnas.
Actualizar los punteros de fila:

Después de procesar cada fila, se actualiza el puntero de fila filas en el resultado con el contador cont, que lleva la cuenta del número total de elementos no nulos procesados hasta el momento.
Devolver el resultado: Se devuelve el diccionario resultado que contiene la suma de las dos matrices en formato CSR.

g. Matriz Transpuesta

   La transpuesta de la matriz, se realiza por separado para cada uno de los formatos, dado que se trabajara basándonos en cada uno de los formatos de salida de manera individual.

   Formato COO:

```
def TransponerCOO(matriz_coo):
    transpuesta = {
        "valores": matriz_coo["valores"],
        "filas": matriz_coo["columnas"],
        "columnas": matriz_coo["filas"]
    }
    return transpuesta
```

   Inicialización de la matriz transpuesta: Se crea un diccionario transpuesta con las listas valores, filas, y columnas.

   Intercambio de filas y columnas: La lista valores se copia directamente de la matriz original. Las listas filas y columnas se intercambian, de modo que las filas de la matriz original se convierten en las columnas de la matriz transpuesta y viceversa.

Devolver la matriz transpuesta: Se devuelve el diccionario transpuesta que contiene la matriz transpuesta en formato COO.

Formato CSC:

```python
def TransponerCSC(matriz_csc):
    num_cols = len(matriz_csc["p_col"]) - 1
    num_filas = max(matriz_csc["fil"]) + 1 if matriz_csc["fil"] else 0

    transpuesta = {
        "val": [],
        "fil": [],
        "p_col": [0] * (num_filas + 1)
    }

    # Contar el número de elementos en cada fila de la matriz original
    for fila in matriz_csc["fil"]:
        transpuesta["p_col"][fila + 1] += 1

    # Convertir los conteos acumulados en punteros de columna
    for i in range(1, len(transpuesta["p_col"])):
        transpuesta["p_col"][i] += transpuesta["p_col"][i - 1]

    # Crear una lista temporal para almacenar los índices de inserción
    inserciones = transpuesta["p_col"][:]

    # Llenar los valores y las filas de la matriz transpuesta
    for col in range(num_cols):
        start = matriz_csc["p_col"][col]
        end = matriz_csc["p_col"][col + 1]
        for i in range(start, end):
            fila = matriz_csc["fil"][i]
            val = matriz_csc["val"][i]
            idx = inserciones[fila]
            transpuesta["val"].insert(idx, val)
            transpuesta["fil"].insert(idx, col)
            inserciones[fila] += 1

    return transpuesta
```

Inicialización de la matriz transpuesta: Se crea un diccionario transpuesta con listas vacías para val, fil, y p_col. La lista p_col se inicializa con ceros y su longitud es igual al número de filas de la matriz original más uno.

Contar el número de elementos en cada fila de la matriz original: Se recorre la lista fil de la matriz original y se incrementa el contador correspondiente en p_col de la matriz transpuesta. Esto cuenta cuántos elementos hay en cada fila de la matriz original, que se convertirán en columnas en la matriz transpuesta.

Convertir los conteos acumulados en punteros de columna: Se convierte la lista p_col en una lista de punteros acumulados, donde cada elemento indica el índice de inicio de una columna en las listas val y fil de la matriz transpuesta.

Crear una lista temporal para almacenar los índices de inserción: Se crea una copia de la lista p_col para llevar la cuenta de los índices de inserción en las listas val y fil de la matriz transpuesta.

Llenar los valores y las filas de la matriz transpuesta: Se recorre cada columna de la matriz original y se insertan los valores y las filas correspondientes en las listas val y fil de la matriz transpuesta, utilizando los índices de inserción almacenados en la lista temporal.

Devolver la matriz transpuesta: Se devuelve el diccionario transpuesta que contiene la matriz transpuesta en formato CSC.

Formato CSR:

```python
def TransponerCSR(matriz_csr):
    num_filas = len(matriz_csr["filas"]) - 1
    num_columnas = max(matriz_csr["columnas"]) + 1 if matriz_csr["columnas"]

    transpuesta = {
        "valores": [],
        "columnas": [],
        "filas": [0] * (num_columnas + 1)
    }

    # Contar el número de elementos en cada columna de la matriz original
    for col in matriz_csr["columnas"]:
        transpuesta["filas"][col + 1] += 1

    # Convertir los conteos acumulados en punteros de fila
    for i in range(1, len(transpuesta["filas"])):
        transpuesta["filas"][i] += transpuesta["filas"][i - 1]

    # Crear una lista temporal para almacenar los índices de inserción
    inserciones = transpuesta["filas"][:]

    # Llenar los valores y las columnas de la matriz transpuesta
    for fila in range(num_filas):
        start = matriz_csr["filas"][fila]
        end = matriz_csr["filas"][fila + 1]
        for i in range(start, end):
            col = matriz_csr["columnas"][i]
            val = matriz_csr["valores"][i]
            idx = inserciones[col]
            transpuesta["valores"].insert(idx, val)
            transpuesta["columnas"].insert(idx, fila)
            inserciones[col] += 1

    return transpuesta
```

Inicialización de la matriz transpuesta: Se crea un diccionario transpuesta con listas vacías para valores, columnas, y filas. La lista filas se inicializa con ceros y su longitud es igual al número de columnas de la matriz original más uno.

Contar el número de elementos en cada columna de la matriz original: Se recorre la lista columnas de la matriz original y se incrementa el contador correspondiente en filas de la matriz transpuesta. Esto cuenta cuántos

elementos hay en cada columna de la matriz original, que se convertirán en filas en la matriz transpuesta.

Convertir los conteos acumulados en punteros de fila: Se convierte la lista filas en una lista de punteros acumulados, donde cada elemento indica el índice de inicio de una fila en las listas valores y columnas de la matriz transpuesta.

Crear una lista temporal para almacenar los índices de inserción: Se crea una copia de la lista filas para llevar la cuenta de los índices de inserción en las listas valores y columnas de la matriz transpuesta.

Llenar los valores y las columnas de la matriz transpuesta: Se recorre cada fila de la matriz original y se insertan los valores y las columnas correspondientes en las listas valores y columnas de la matriz transpuesta, utilizando los índices de inserción almacenados en la lista temporal.

Devolver la matriz transpuesta: Se devuelve el diccionario transpuesta que contiene la matriz transpuesta en formato CSR.

BIBLIOGRAFÍA Y REFERENCIAS

[1]. https://rramosp.github.io/introalgs.v1/content/NOTAS%2003%20-%20Matrices%20dispersas.html

[2]. https://www.youtube.com/watch?v=kzJ9Ux2xwSI

[3]. https://www.delftstack.com/es/howto/python/sparse-matrix-in-python/