



## EJERCICIO DE MATRIZ Y LISTA DE ADYACENCIA

### **Presenta**

Cristian David Mora Sáenz

### **Docente**

Segundo Fidel Puerto Garavito

### **Asignatura**

Diseño de Algoritmos

**NRC:** 7487

Corporación Universitaria Minuto de Dios

Facultad de Ingeniería

Colombia, Bogotá

abril 28 de 2020.

Las matrices y listas de adyacencia se utilizan para conocer los “vecinos” o nodos que están conectados en un grafo entre sí, esto con el fin de saber los pesos de las aristas.

Primero se implementó un ejemplo de listas adyacentes, con nodos ya definidos, de la siguiente forma:

1. Primero se inicializaron los nodos los cuales están unidos por una variable denominada “links”, con ayuda del arreglo links, se puede almacenar con que nodos se está conectando un nodo principal. Esto se realiza mediante un ciclo FOR, para recorrer el arreglo que representa los valores del grafo.

```
observaciones = [(20, 1), (26, 1), (12, 2), (14, 2), (15,3 ), (14, 3), (10, 3)]  
  
Links = {}  
  
for (eventid, mnbr) in observaciones :  
    if not eventid in Links.keys():  
        Links[eventid] = []  
  
        Links[eventid].append(mnbr)  
  
mnbrs = set([mnbr for (eventid, mnbr) in observaciones])
```

2. Por último, se ordenan los nodos en sentido ascendente, debido a que la representación de los nodos se encuentra determinada numéricamente.

```
mnbrLinks = { mnbr : set() for mnbr in mnbrs }  
  
for mnbrList in Links.values() :  
    for mnbr in mnbrList:  
        mnbrLinks[mnbr] = mnbrLinks[mnbr].union(set(mnbrList))
```

3. Finalmente, se imprime una lista con los nodos y sus respectivas aristas.

```
print(mnbrLinks)
```

Luego, se ejecuta el código del archivo "listaAdyacente.py" y se obtiene como resultado: Teniendo en cuenta que los números separados por coma y que se encuentran dentro de los corchetes, son el número de aristas que posee.

```
PS C:\Users\CristianW\Desktop\Ejercicios de Diseño> & C:/Users/CristianW/AppData/Local/Programs/Python/Python38-32/python.exe "c:/Users/CristianW/Desktop/Matris de Adyacencia/ListaAdyacente.py"
{1: {1}, 2: {2, 3}, 3: {2, 3}}
```

## MATRIZ DE ADYACENCIA

La matriz de adyacencia se utiliza para encontrar las relaciones binarias entre nodos.

1. La matriz de adyacencia muestra las relaciones entre los nodos, teniendo en cuenta la línea principal de la matriz.

CÓDIGO:

```
import os

adjacencyList = {1: {1}, 2: {2, 3}, 3: {2, 3}}

class AdjacencyMatrix():

    def __init__(self, adjacencyList, label = ""):
        """
        Instanciation method of the class.
        Create an adjacency matrix from an adjacencyList.
        It is supposed that graph vertices are labeled with numbers from 1 to n.
        """

        self.matrix = []
        self.label = label

        #create an empty matrix
        for i in range(len(adjacencyList.keys())):
            self.matrix.append( [0]*(len(adjacencyList.keys())) )

        for key in adjacencyList.keys():
            for value in adjacencyList[key]:
                self[key-1][value-1] = 1
```

```

def __str__(self):
    # return self.__repr__() is another possibility that just print the list of list
    # see python doc about difference between __str__ and __repr__

    #label first line
    string = self.label + "\n"
    for i in range(len(self.matrix)):
        string += str(i+1) + "\t"
    string += "\n"

    #for each matrix line :
    for row in range(len(self.matrix)):
        string += str(row+1) + "\t"
        for column in range(len(self.matrix)):
            string += str(self.matrix[row][column]) + "\t"
        string += "\n"

    return string

def __repr__(self):
    return str(self.matrix)

def __getitem__(self, index):
    """ Allow to access matrix element using matrix[index][index] syntax """
    return self.matrix.__getitem__(index)

def __setitem__(self, index, item):
    """ Allow to set matrix element using matrix[index][index] = value syntax """
    return self.matrix.__setitem__(index, item)

def areAdjacent(self, i, j):
    return self[i-1][j-1] == 1

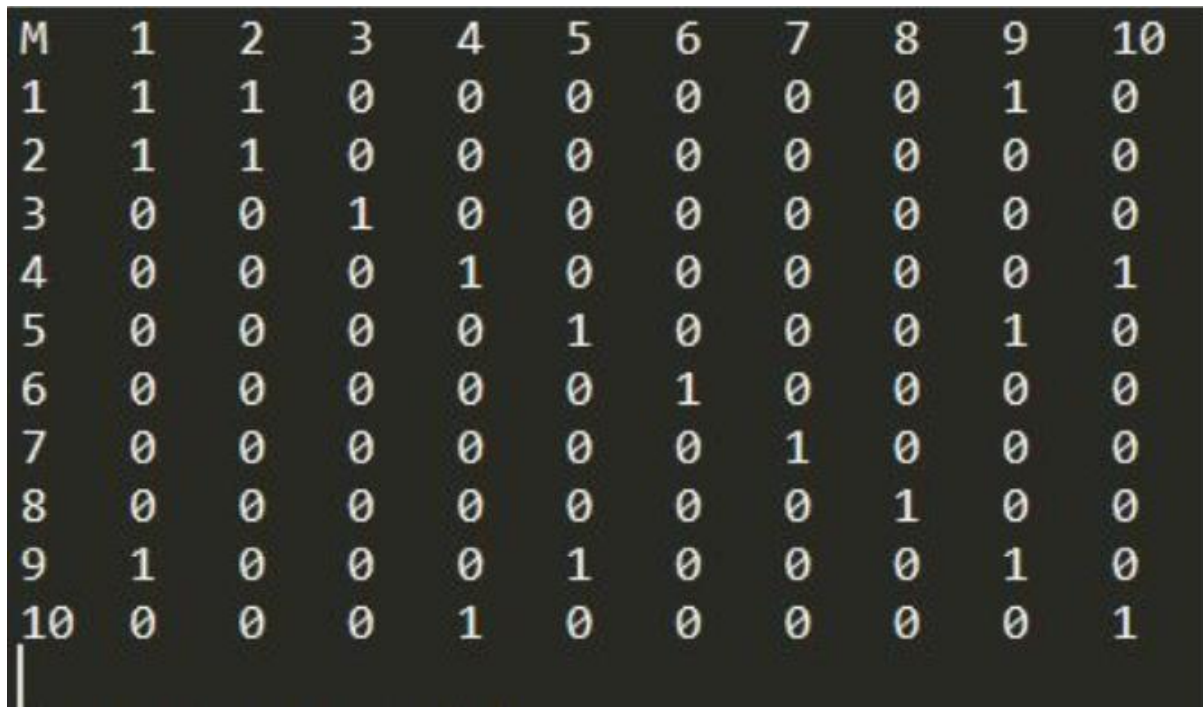
```

```

m = AdjacencyMatrix(adjacencyList, label="mbr")
print(m)
print("m.areAdjacent(1,2) :",m.areAdjacent(1,2))
print("m.areAdjacent(2,3) :",m.areAdjacent(2,3))
file = open("C:/Users/Acer/Desktop/Matriz y lista adyacente/MatrizAdyacente.txt", "w")
m.write("m.areAdjacent(1,2) :",m.areAdjacent(1,2)+ os.linesep)
m.write("m.areAdjacent(2,3) :",m.areAdjacent(2,3))
file.close()

```

Final mente la siguiente imagen representa la matriz de adyacencia que da como resultados de nuestro código.

A 10x10 adjacency matrix displayed on a black background with yellow text. The matrix is symmetric, indicating an undirected graph. The nodes are labeled 1 through 10. The diagonal elements are all 1, representing self-loops. The off-diagonal elements are 1 for the following pairs of nodes: (1,2), (2,1), (2,3), (3,2), (4,5), (5,4), (6,7), (7,6), (8,9), (9,8), (9,10), and (10,9). All other elements are 0.

M	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	0	0	0	0	1	0
2	1	1	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	1
5	0	0	0	0	1	0	0	0	1	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	1	0	0
9	1	0	0	0	1	0	0	0	1	0
10	0	0	0	1	0	0	0	0	0	1

Enlace a GitHub: <https://github.com/CristianMoraS/Matriz-de-Adyacencia>