En el siguiente reporte hablaremos de los 5 algoritmos vistos en clase: fila, pila, grafo, bfs, dfs.

Fila:

Este código se trata de poder ingresar valores de modo que los acomode en un espacio de memoria por fila de manera que cuando quiera acceder a los valores de mencionada fila se realice de uno por uno empezando del primer hasta el último punto

```
class Fila:


    def __init__(self):
        self.fila = []
    def obtener(self):
        return self.fila.pop(0)
    def meter(self,e):
        self.fila.append(e)
        return len(self.fila)
    @property
    def longitud(self):
        return len(self.fila)
```

Pila:

Pila trabaja de manera similar a “Fila” con una diferencia en donde este trabaja de manera inversa, cuando fila te mostraba los elementos del primer elemento hasta el último en pila los muestra comenzando desde el último elemento hasta llegar al primer elemento ingresado.

```
class Pila:
    def __init__(self):
        self.pila = []
    def obtener(self):
        return self.pila.pop()
    def meter(self,e):
        self.pila.append(e)
        return len(self.pila)
    @property
    def longitud(self):
        return len(self.pila)
```

Grafo:

Este programa trabaja con un conjunto de objetos llamado vértices o nodos unidos por aristas o arcos y así permiten mostrar las combinaciones binarias entre elementos de un conjunto.

A nosotros nos tocó después de introducir el código ingresar los datos de un grafo ya realizado por nosotros para así poder ver como trabajaba.

```
class Grafo:
    def __init__(self):
        self.V = set() #un conjunto
        self.E = dict() #un mapeo de pesos de aristas
        self.vecinos = dict() # un mapeo
    def agregar(self, v):
        self.V.add(v)
        if not v in self.vecinos: #vecindad de v
            self.vecinos[v] = set() #inicialmente no tiene nada
    def conecta(self, v, u, peso=1):
        self.agregar(v)
        self.agregar(u)
        self.E[(v, u)] = self.E[(u, v)] = peso  # en ambos sentidos
        self.vecinos[v].add(u)
        self.vecinos[u].add(v)
g= Grafo()
g.conecta('Tamara', 'Pilar', 1)
g.conecta('Tamara', 'Alex', 1)
g.conecta('Tamara', 'Mango', 1)
g.conecta('Tamara', 'Evelyn', 1)
g.conecta('Tamara', 'Sarahí', 1)
g.conecta('Pilar', 'Alex', 1)
g.conecta('Pilar', 'Evelyn', 1)
g.conecta('Alex', 'Evelyn', 1)
g.conecta('Alex', 'Ismael', 1)
g.conecta('Alex', 'Mango', 1)
g.conecta('Mango', 'Sarahí', 1)
g.conecta('Sarahí', 'Ismael', 1)
g.conecta('Ismael', 'Evelyn', 1)
g.conecta('Evelyn', 'Rafael', 1)

print(BFS(g, 'Tamara'))
print(BFS(g, 'Evelyn'))
print(BFS(g, 'Rafael'))
```

BFS:

Este código trabaja a la par con los algoritmos de grafo y fila donde vendría a buscar un cierto elemento dentro de cierto grafo definido de principio a fin.

```
def BFS(g, ni):
    visitados = []
```

```
    f= Fila()
    f.meter(ni)
    while (f.longitud > 0):
        na = f.obtener()
        if na not in visitados:
            visitados.append(na)
            ln = g.vecinos[na]
            for nodo in ln:
                if nodo not in visitados:
                    f.meter(nodo)
    return visitados
```

```
import random
    V1=random.choice(list(g.V))
```

DFS:

En este código se trabaja también con grafo pero en lugar de fila, es pila, donde también busca un elemento del grafo comenzando desde el final.

```
def DFS(g, ni):
    visitados =[]
    f= Pila()
    f.meter(ni)
    while(f.longitud > 0):
        na = f.obtener()
        if  na not in visitados:
            visitados.append(na)
            ln = g.vecinos[na]
            for nodo in ln:
                if nodo not in visitados:
                    f.meter(nodo)
    return visitados
```

Cuando nosotros definimos los elementos del grafo que habíamos llevado con un código llamamos a una biblioteca para que nos diera un elemento del grafo al azar y así ver cual eran sus elementos, a continuación también se muestra los resultados que arrojo Grafo

>>> import random

>>> V1=random.choice(list(g.V))

>>> V1

'Rafael'

RESULTADOS DE GRAFO

```
>> print(BFS(g, 'Tamara'))
['Tamara', 'Alex', 'Sarahí', 'Mango', 'Pilar', 'Evelyn', 'Mango', 'Ismael', 'Pilar', 'Evelyn
', 'Ismael', 'Mango', 'Evelyn', 'Rafael', 'Ismael', 'Rafael', 'Rafael']
>>> print(BFS(g, 'Evelyn'))
['Evelyn', 'Alex', 'Rafael', 'Ismael', 'Tamara', 'Pilar', 'Mango', 'Ismael', 'Pilar', 'Tamar
a', 'Sarahí', 'Sarahí', 'Mango', 'Pilar', 'Sarahí', 'Sarahí', 'Sarahí']
>>> print(BFS(g, 'Rafael'))
['Rafael', 'Evelyn', 'Alex', 'Ismael', 'Tamara', 'Pilar', 'Mango', 'Ismael', 'Pilar', 'Tamar
a', 'Sarahí', 'Sarahí', 'Mango', 'Pilar', 'Sarahí', 'Sarahí', 'Sarahí']
>>>
>>> print(g.V)
{'Alex', 'Sarahí', 'Mango', 'Ismael', 'Rafael', 'Pilar', 'Tamara', 'Evelyn'}
>>> print(g.E)
{('Tamara', 'Pilar'): 1, ('Pilar', 'Tamara'): 1, ('Tamara', 'Alex'): 1, ('Alex', 'Tamara'):
1, ('Tamara', 'Mango'): 1, ('Mango', 'Tamara'): 1, ('Tamara', 'Evelyn'): 1, ('Evelyn', 'Tama
ra'): 1, ('Tamara', 'Sarahí'): 1, ('Sarahí', 'Tamara'): 1, ('Pilar', 'Alex'): 1, ('Alex', 'P
ilar'): 1, ('Pilar', 'Evelyn'): 1, ('Evelyn', 'Pilar'): 1, ('Alex', 'Evelyn'): 1, ('Evelyn',
 'Alex'): 1, ('Alex', 'Ismael'): 1, ('Ismael', 'Alex'): 1, ('Alex', 'Mango'): 1, ('Mango', '
Alex'): 1, ('Mango', 'Sarahí'): 1, ('Sarahí', 'Mango'): 1, ('Sarahí', 'Ismael'): 1, ('Ismael
', 'Sarahí'): 1, ('Ismael', 'Evelyn'): 1, ('Evelyn', 'Ismael'): 1, ('Evelyn', 'Rafael'): 1,
('Rafael', 'Evelyn'): 1}
```