

Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

## **Evaluación 2**

Matemáticas Computacionales.

Maestro

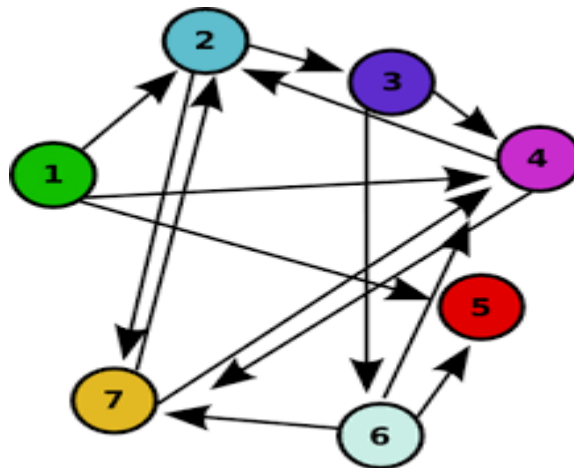
José Anastasio Hernández Saldaña.

Alumno

Leslie Jaressy Morales Ortiz

Matricula: 1661971

Grupo y Salón: 112



San Nicolás, Ciudad Universitaria, a 06 de Octubre de 2017

# EVALUACION 2.

Acontinuacion se presentan una pequeña descripcion de lo que estuvimos trabajando a lo largo de esta evaluacion , asi mismo el codigo final donde implementamos todos los temas vistos.

## 1. Fila.

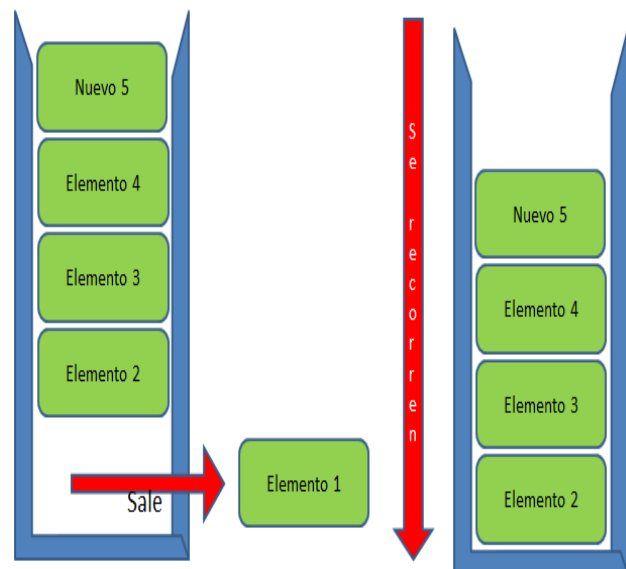
*Def.*

Es una estructura de datos que permite almacenar datos en el orden FIFO (First In First Out) en español, Primero en Entrar, Primero en Salir). La recuperación de los datos es hecha en el orden en que son insertados.

Apartir de un arreglo realiza la funcion de regresar el primer elemento que ingresaste ,su manera de trabajar me resulta muy interesante debido a que despues de arrojarle a dicho elemento éste deja de ser el primero y cuando nuevamente se vuelva a llamar a la funcion fila tomara al siguiente elemento y actua de la misma manera.

```
class Fila(object):
    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)
```

```
fila=Fila()
fila.meter(5)
fila.meter('fila')
fila.meter("leslie")
print(fila.longitud)
print(fila.obtener())
print(fila.obtener())
print(fila.obtener())
print(fila.longitud)
```



## 2. Pila.

*Def.*

Una pila representa una estructura lineal de datos en que se puede agregar o quitar elementos únicamente por uno de los dos extremos. En consecuencia, los elementos de una pila se eliminan en el orden inverso al que se insertaron.

A simple vista los codigos de pila y fila pueden parecer casi iguales porque en realidad lo son , es decir se necesito de solo cambiar algunas variables para crear pila,aunque no debemos confundirnos porque la manera de trabajar de cada una de estas estructuras es variada; en esté trabajamos con la declaracion de un arreglo donde la estructura pila al ser ejecutada regresa el último elemento que fue ingresado en el arreglo .

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

```
pila=Pila()
pila.meter(4)
pila.meter('a')
pila.meter("leslie")
print(pila.longitud)
print(pila.obtener())
print(pila.obtener())
print(pila.obtener())
print(pila.longitud)|
```



### 3. Grafo.

*Def.*

Los grafos no son más que la versión general de un árbol, es decir, cualquier nodo de un grafo puede apuntar a cualquier otro nodo de éste (incluso a él mismo). Los grafos se usan para almacenar datos que están relacionados de alguna manera (relaciones de parentesco, puestos de trabajo, ...); por esta razón se puede decir que los grafos representan la estructura real de un problema.

Como hace mención la def. nosotros trabajamos con un nodo principal y a partir de éste comenzamos a conectar a otros, incluso puede ocurrir que existan nodos que no están conectados al principal, cabe destacar que este enlace se realiza gracias al uso de aristas que funcionan como cuerdas de conexión entre cada uno.

Otra de las cosas que es necesario mencionar es el grafo con peso que de alguna manera nos sirve como longitud entre cada uno de ellos.

#### **Terminología de grafos:**

- **Vértice** : Nodo.
- **Enlace** : Conexión entre dos vértices (nodos).
- **Adyacencia** : Se dice que dos vértices son adyacentes si entre ellos hay un enlace directo.
- **Vecindad** : Conjunto de vértices adyacentes a otro.
- **Camino** : Conjunto de vértices que hay que recorrer para llegar desde un nodo origen hasta un nodo destino.
- **Grafo conectado** : Aquél que tiene camino directo entre todos los nodos.
- **Grafo dirigido** : Aquél cuyos enlaces son unidireccionales e indican hacia donde están dirigidos.
- **Grafo con pesos** : Aquél cuyos enlaces tienen asociado un valor. En general en este tipo de grafos no suele tener sentido que un nodo se apunte a sí mismo porque el coste de este enlace sería nulo.

```

class Grafo:
    def __init__(self):
        self.CTO = set() # un conjunto
        self.MARI = dict() # un mapeo de pesos de aristas
        self.vec = dict() # un mapeo

    def agrega(self, v):
        self.CTO.add(v)
        if not v in self.vec: # vecindad de v
            self.vec[v] = set() # inicialmente no tiene nada

    def conecta(self, v, u, peso=1):
        self.agrega(v)
        self.agrega(u)
        self.MARI[(v, u)] = self.MARI[(u, v)] = peso # en ambos sentidos
        self.vec[v].add(u)
        self.vec[u].add(v)

    @property
    def complemento(self):
        comp=Grafo()
        for v in self.ver:
            for w in self.ver:
                if v!=w and (v,w) not in self.ARI:
                    comp.conecta(v,w,1)
        return comp

```

## 4. BFS (Breadth First Search – Búsqueda por Anchura)

*¿Como trabaja?*

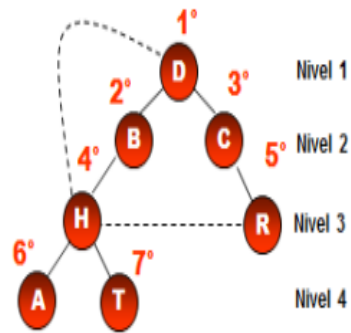
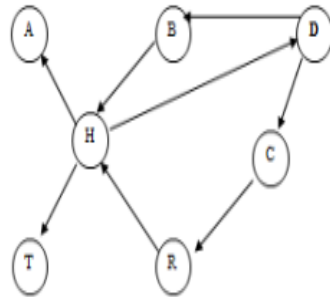
Todo parte de un nodo inicial que será la raíz luego va a los adyacentes a ese nodo y los agrega en una cola, como la prioridad de una cola es FIFO (primero en entrar es el primero en salir), los siguientes nodos a evaluar serán los adyacentes previamente insertados. una cosa bastante importante es el hecho de que no se pueden visitar 2 veces el mismo nodo o Estado. ya que si no podríamos terminar en un ciclo interminable o simplemente no hallar el punto deseado en el menor número de pasos.

En lo personal yo lo visualizo como un trabajo por niveles donde si en lo absoluto parte del nodo principal y a partir de aquí visita cada respectivo nivel.

## Implementación

- ✓ Para encontrar la ruta más corta cuando el peso entre todos los nodos es 1.
- ✓ Cuando se requiere llegar con un movimiento de caballo de un punto a otro con el menor número de pasos.
- ✓ Cuando se desea transformar algo un numero o cadena en otro realizando ciertas operaciones como suma producto.
- ✓ O para salir de un laberinto con el menor número de pasos, etc. Podrán aprender a identificarlos con la práctica.

```
def BFS(self, ni):  
    visitados = []  
    f=Fila()  
    f.meter(ni)  
    while(f.longitud>0):  
        na =f.obtener()  
        visitados.append(na)  
        ln = self.vec[na]  
        for nodo in ln:  
            if nodo not in visitados:  
                f.meter(nodo)  
    return visitados
```



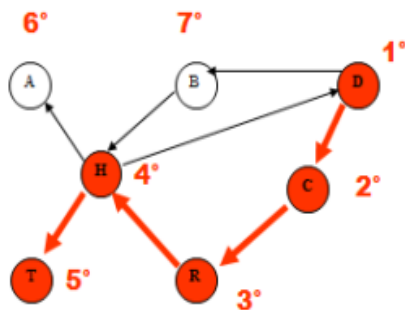
Recorrido desde Vertice por anchura desde vertice D = {D, B, C, H, R, A, T}

## 5.DFS (Deaph First Search – Busqueda por profundidad)

Es equivalente a un recorrido en preorden de un árbol. Se elige un nodo v de partida. Se marca como visitado y se recorren los nodos no visitados adyacentes a v, usando recursivamente la búsqueda primero en profundidad.

Éste en lo peculiar trabaja preguntándose si un nodo tiene o no hijos, vea la imagen a continuación donde partimos del nodo D el cual tiene 2 hijos y se inclina por ir a visitar a C, después C tiene de hijo a R y este a su vez a H quien tiene 2 hijos A y T pero opta por visitar T, posteriormente A ,por ultimo regresa a D donde partió y recorre al otro hijo quien es B.

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



Recorrido por profundidad desde Vértice D= {D, C, R, H, T, A, B}

## Codigo.

```
class Fila(object):
    def __init__(self):
        self.fl=[]
    def obtener(self):
        return self.fl.pop(0)
    def meter(self,e):
        self.fl.append(e)
        return len(self.fl)
    @property
    def longitud(self):
        return len(self.fl)
class Pila(object):
    def __init__(self):
        self.pl=[]
    def obtener(self):
        return self.pl.pop()
    def meter(self,e):
        self.pl.append(e)
        return len(self.pl)
    @property
    def longitud(self):
        return len(self.pl)
class Grafo:

    def __init__(self):
        self.CTO = set() # un conjunto
        self.MARI = dict() # un mapeo de pesos de aristas
        self.vec = dict() # un mapeo

    def agrega(self, v):
        self.CTO.add(v)
        if not v in self.vec: # vecindad de v
            self.vec[v] = set() # inicialmente no tiene nada

    def conecta(self, v, u, peso=1):
        self.agrega(v)
        self.agrega(u)
        self.MARI[(v, u)] = self.MARI[(u, v)] = peso # en ambos sentidos
        self.vec[v].add(u)
        self.vec[u].add(v)

    @property
    def complemento(self):
        comp=Grafo()
        for v in self.ver:
            for w in self.ver:
                if v!=w and (v,w) not in self.ARI:
                    comp.conecta(v,w,1)
        return comp
```



```

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

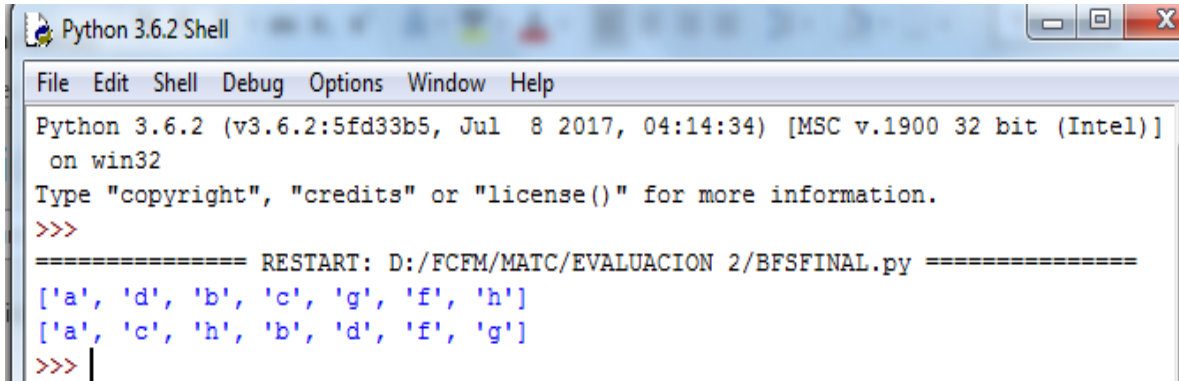
def BFS(self,ni):
    visitados =[]
    f=Fila()
    f.meter(ni)
    while(f.longitud>0):
        na =f.obtener()
        visitados.append(na)
        ln = self.vec[na]
        for nodo in ln:
            if nodo not in visitados:
                f.meter(nodo)
    return visitados

g=Grafo()
g.conecta('a','b')
g.conecta('a','c')
g.conecta('a','d')
g.conecta('d','f')
g.conecta('d','g')
g.conecta('c','h')

print(g.BFS('a'))
print(g.DFS('a'))

```

## Resultados.



```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/FCFM/MATC/EVALUACION 2/BFSFINAL.py =====
['a', 'd', 'b', 'c', 'g', 'f', 'h']
['a', 'c', 'h', 'b', 'd', 'f', 'g']
>>> |
```

## Conclusión.

En esta segunda evaluación trabajamos con estructuras de datos, grafos, búsquedas, se podría decir que comenzamos con lo más simple, a decir verdad, fue de suma importancia el apoyo del profesor debido a que en su mayoría él nos proporcionó las herramientas para la realización de cada una de las actividades que hoy conforman este trabajo.

Comenzare por nombrar a cada uno de los conceptos con los que se trabajó, de principio hablemos de filas y pilas las cuales agrupé en una sola categoría debido a que yo encuentro en estas estructuras cosas similares, de principio la elaboración del código, la entrada de datos; así mismo me quedo con la enseñanza de que, en la primera realizara su proceso y arrojara a el elemento más viejo del arreglo o en otras palabras aquel que tomó la posición número 1; El segundo lo relaciono a partir de una entrada de datos donde ejecuta y me muestra a el elemento más nuevo del arreglo, me refiero a ese que fue introducido al final.

Por otra parte, el grafo es la base para la implementación de las búsquedas ya sea por amplitud o profundidad, de momento solo trabajamos con un grafo sencillo donde ingresamos los elementos en un arreglo, posteriormente las búsquedas realizaban su trayecto y me mostraban en pantalla el orden final, desde mi punto de vista me parece que esto es tan solo la base para la realización de grandes proyectos en la gran variedad de áreas que están a nuestro alcance como la optimización.

Yo como estudiante en la licenciatura de matemáticas pienso que en ocasiones no se les da la importancia debida a algunas materias, pero con el paso del semestre me he dado cuenta de muchas cosas porque he de confesar que comencé creyendo que matemáticas computacionales sería una materia tediosa pero hoy me doy cuenta de que materias como estas son en empuje que necesita dicha carrera para entrar a lo grande en la era tecnológica.