UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FCFM
FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

Matemáticas Computacionales
Reporte de Grafos

Brandon Rodrigo Ceballos Salazar
Matrícula 1724045

Fila

Una fila en una estructura de datos en la cual cada elemento ingresado tiene que "entrar" por la parte inferior y si algún elemento quiere salir tiene que salir de la parte superior.

En Python se puede representar como:

```
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

Una pila (stack en inglés) es una lista ordenada o estructura de datos que permite almacenar y recuperar datos, el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In, First Out, «último en entrar, primero en salir»).

En Python se puede representar como:

```
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

Un grafo en el ámbito de las ciencias de la computación es un tipo abstracto de datos, que consiste en un conjunto de nodos y un conjunto de aristas que establecen relaciones entre los nodos.

En Python se puede representar como:

```
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 agrega(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.agrega(v)
        self.agrega(u)
        self.E[(v,  u)] = self.E[(u,  v)] = peso # en ambos sentidos
        self.vecinos[v].add(u)
        self.vecinos[u].add(v)
    def complemento(self):
        comp= Grafo()
        for v in self.V:
```

```
        for w in self.V:
            if v != w and (v, w) not in self.E:
                comp.conecta(v,  w, 1)
    return comp
```

Búsqueda por profundidad

Es un algoritmo de búsqueda no informada utilizado para recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

En Python se puede representar como:

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

Búsqueda por profundidad

Es un algoritmo de búsqueda no informada utilizado para recorrer o buscar elementos en un grafo. Intuitivamente, se comienza en la raíz y se exploran todos los vecinos de este nodo. A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.

En Python se puede representar como:

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

```
            f.meter(nodo)
    return visitados
```

Promedio

En un grafo, el promedio está denotado por la suma de los grados divididos entre el número de vértices.

En Python se puede representar como:

```
def promedio(self):
        grados = []
        for v in self.V:
                grados.append(len(self.vecinos[v]))
        return sum(grados)/len(self.V)
```

Densidad

En un grafo, la densidad está denotada por la cantidad de aristas entre la cantidad máxima de aristas.

En Python se puede representar como:

```
def densidad(self):
        cantidadAristas= len(self.E)
        cantidadVertices= len(self.V)
        cantidadMaximaAristas= cantidadVertices*(cantidadVertices - 1)
        if cantidadMaximaAristas < 1:
                cantidadAristas=1
        dens= cantidadAristas/cantidadMaximaAristas
        return dens
```

Radio

El radio de un grafo es la excentricidad mínima de todos los vértices pertenecientes al grafo. La excentricidad de un vértice es la distancia máxima entre todos los vértices pertenecientes al grafo.

En Python se puede representar como:

```
@property
        def radio(self):
        di_ma=dict() #distancias maximas de cada vertice
        for v in self.V:
                diccionario = BFS_N(self, v)
                di_ma[v]= max(diccionario.values() )
        radio = min(di_ma.values())
        return radio
```

Centro

Conocido el radio, se seleccionan los vértices cuya excentricidad sea dicho radio y las correspondientes aristas

En Python se puede representar como:

```
@property
    def centrales(self):
        di_ma=dict() #distancias maximas de cada vertice
        nodos_centrales=[]
        for v in self.V:
            diccionario = BFS_N(self, v)
            di_ma[v]= max(diccionario.values() )
        radio = min(di_ma.values())
        for valor in di_ma:
            if di_ma[valor] == radio:
            nodos_centrales.append(valor)
        return nodos_centrales
```

Diámetro

A diferencia del radio, el diámetro es la mayor de las excentricidades entre todos los vértices del grafo.

En Python se puede representar como:

```
@property
    def diametro(self):
    maximo = 0
    for vertice in self.V:
        dic_bfs = BFS_N(self, vertice)
        if max( dic_bfs.values() )>maximo:
    maximo = max(dic_bfs.values())
    return maximo
```