

Grafo, Pila y Fila

Facultad de ciencias Físico - Matemáticas

Universidad Autonoma de Nuevo León

Sarai Elisabet Gómez Ibarra

1748263

Matemáticas computacionales

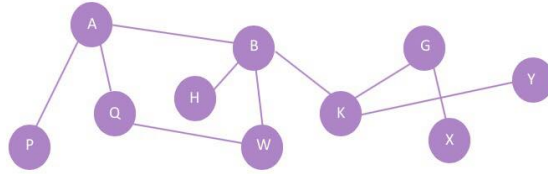
Abril 2018

1 Grafo

Un grafo es una representación gráfica de diversos puntos que se conocen como nodos o vértices, los cuales se encuentran unidos a través de líneas que reciben el nombre de aristas. Al analizar los grafos, los expertos logran conocer cómo se desarrollan las relaciones recíprocas entre aquellas unidades que mantienen algún tipo de interacción en otras palabras un grafo es un conjunto que está compuesto por vértices y aristas, estos vértices tienen un valor y están unidos con los demás vértices por las aristas, que de igual manera tienen un valor numérico. Los vértices los podemos tomar como personas que están en su lugar fijos, y las aristas es la ruta para llegar con tu vecino.

En Python el grafo está programado 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)
```



2 Pila

Una pila una lista ordenada o estructura de datos que permite almacenar y recuperar datos en el cual cada elemento que entra tiene que ingresar por la parte superior y si algún elemento quiere salir tiene que salir de la parte superior. Un ejemplo sería al momento de que una persona está apilando cajas pesadas, si quiere sacar alguna tiene que quitar la caja que esta hasta la cima y así sucesivamente hasta llegar a la caja deseada.

En Python la pila está programado 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)
```

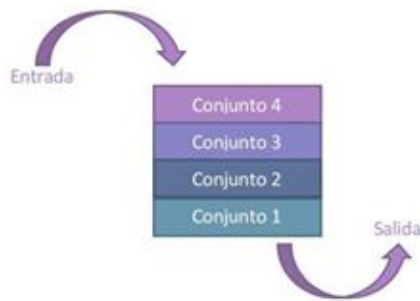


3 Fila

Una fila es una forma de ordenamiento en el cual cada elemento que entra tiene que ingresar por la parte inferior y si algún elemento quiere salir tiene que salir de la parte superior. Un ejemplo sería al momento de que una persona se forma para que le proporcionen algún servicio, esta persona se pone al final de la fila y el sujeto de este dando el servicio va a estar atendiendo a cada persona del primero que este en la fila, así sucesivamente hasta llegar a la persona de ingreso al final.

En Python la fila está programado 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)
```



4 Búsqueda por profundidad

En el grafo tomamos a un vértice como base y de ese podemos ver cómo podemos visitar a cada uno de los vecinos, sin tener que visitarlo más de una ocasión esto sería con ayuda del programa DFS, establecido en clase. El programa DFS función gracias al programa Pila() visto anteriormente.

El programa DFS en Python está dado por:

```

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

```

5 Búsqueda por amplitud

En el grafo se toma un nodo inicial al cual se le denomina raíz, si así se le denomina en estructura de datos; luego desde la raíz se visitan todos los nodos por niveles partiendo de la izquierda luego derecha, una vez que se termina con el segundo nivel se pasa al tercero y así sucesivamente hasta recorrer todo el árbol o grafo, esto sería con ayuda del programa BFS, establecido en clase. El programa BFS función gracias al programa Fila() visto anteriormente. El programa BFS en Python está dado por:

```

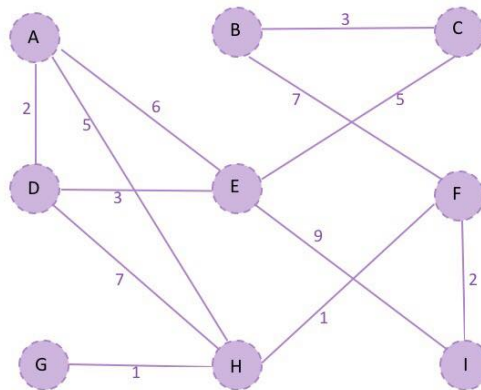
def BFS(g,ni):
    visitados =[]
    f= Fila()
    f.meter(ni)
    while(f.longitud>0):
        na =f.obtener()
        if na not in visitados:
            visitados.append(na)
            ln = g.vecinos[na]
            for nodo in ln:
                if nodo not in visitados:
                    f.meter(nodo)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

```

6 Aplicando el programa

Ahora vamos a dar un ejemplo de como funciona el programa, para esto se usara una funcion en Python la cual elige algun elemento aleatorio, para ver como funciona el programa cuando inicia en un diferente vertice.

Para esto vamos a utilizar el siguiente grafo:



Este grafo se programa en Python de la siguiente manera:

```
g= Grafo()
g.conecta('a','d',2)
g.conecta('a','h',5)
g.conecta('a','e',6)
g.conecta('b','c',7)
g.conecta('b','f',7)
g.conecta('c','e',5)
g.conecta('d','e',3)
g.conecta('d','h',7)
g.conecta('e','i',9)
g.conecta('f','h',1)
g.conecta('f','i',2)
g.conecta('g','h',1)
```

Aplicando el grafo visto anterior en el programa nos da:

```
>>> x= random.choice(list(g.V))
>>> x
' h '
>>> print(BFS(g,'h'))
[ ' h ' , ' g ' , ' f ' , ' d ' , ' a ' , ' b ' , ' i ' , ' e ' , ' c ' ]
>>> print(DFS(g,'h'))
[ ' h ' , ' a ' , ' d ' , ' e ' , ' c ' , ' b ' , ' f ' , ' i ' , ' g ' ]

>>> x= random.choice(list(g.V))
>>> x
' d '
```

```

>>> print(BFS(g,'d'))
[ ' d ' , ' e ' , ' h ' , ' a ' , ' i ' , ' c ' , ' g ' , ' f ' , ' b ' ]
>>> print(DFS(g,'d'))
[ ' d ' , ' a ' , ' h ' , ' f ' , ' i ' , ' e ' , ' c ' , ' b ' , ' g ' ]

>>> x= random.choice(list(g.V))
>>> x
' a '
>>> print(BFS(g,'a'))
[ ' a ' , ' e ' , ' h ' , ' d ' , ' i ' , ' c ' , ' g ' , ' f ' , ' b ' ]
>>> print(DFS(g,'a'))
[ ' a ' , ' d ' , ' h ' , ' f ' , ' i ' , ' e ' , ' c ' , ' b ' , ' g ' ]

```