Práctica 2

Jose David Martinez Ruiz

5 de marzo de 2018

1. Introducción

En esta tarea lo que se busca es mejorar la tarea de manera que ahora los grafos sean creados a partir de una clase y que también se tenga la opción de crear grafos ponderados, grafos dirigidos o ambos. Se desea que la clase sea lo suficientemente flexible para que dentro de ella se puedan hacer esos tipos de grafos.

2. Desarrollo

se crearon funciones adecuadas para poder asi realizar tareas que puedan llegar a hacer falta, una de ellas es la distancia entre dos puntos y la otra es el punto medio entre dos puntos.

```
def distancia(p1, p2):
    return(((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)**(1/2))

def medio(p1, p2):
    return(((p1[0] + p2[0])/2), ((p1[1] + p2[1])/2))
```

Debido a que en la anterior tarea el grafo que se generaba era un grafo simple y la forma en la que se hacia era directa, por así decirlo, lo primero que se tuvo que hacer fue crear una clase con la cual vamos a trabajar. El constructor de esta clase inicializa todas las listas que se van o se pueden utilizar al igual como los diccionarios.

```
def __init__(self):
        self.n = 0
        self.x = dict()
        self.y = dict()
        self.r = dict()
        self.c = dict()
        self.A = []
        self.P = []
        self.D = []
        self.aux = []
```

Luego de eso se creó una función en la clase con la cual todos los nodos fueran creados a partir de la cantidad que sea introducida al igual que algunas de sus características.

```
def crear(self, n):
        self.n = n
        for nodo in range(self.n):
            self.x[nodo] = random()
            self.y[nodo] = random()
            self.r[nodo] = random() + 2
            self.c[nodo] = 0
```

Ya con todos los nodos inicializados se creó otra función encargada de hacer las conexiones entre los nodos. La función recibe como parámetro la probabilidad que utilizará el criterio de conexión,

si las conexiones serán ponderadas o no y si las conexiones serán dirigidas o no.

Se procede de manera similar a la tarea anterior, ahora se mejoró para que no se hiciera el criterio de un nodo con sí mismo y también para que no haya una conexión doble entre los pares de nodos que el criterio conectó, es decir que, si hay una conexión entre un nodo i y un nodo j, no haya una conexión entre el nodo j y el nodo i. La forma en la que se hizo este se muestra a continuación.

```
def conexiones(self, prob, p = 0, d = 0):
        t = 0
        k = 0
        for i in range(self.n):

            for j in range(self.n):
                if i is not j:

                    if abs(self.r[i] - self.r[j])/
                    (distancia((self.x[i], self.y[i]),
                    (self.x[j], self.y[j])))**2 < prob:
                        test = 0
                        if len(self.A) is not 0 or k is 0:
                            for r in range(len(self.A)):
                                if (self.x[i] is self.A[r][2] and
                                self.y[i] is self.A[r][3] and
                                self.x[j] is self.A[r][0] and
                                self.y[j] is self.A[r][1]):
                                    test = test +1
                        if test is 0:
```

La variable k se utiliza como un auxiliar a la hora de verificar que no haya conexiones dobles de manera que cuando se haga la primera conexión el par de nodos los guarde directamente, pero que cuando ya haya elementos en la lista donde están las conexiones verifique que el par de nodos que cumplieron con el criterio no se encuentre de nuevo dentro de la lista de conexiones. Si llega a haber alguna conexión repetida se contabiliza de manera que no se guarde posteriormente.

Una vez que el criterio de conexión se cumpla y que se haya verificado que no hay conexiones repetidas, se guarda el par de nodos en la lsita de conexiones. Ya cuando se haya cuardado el par de nodos se verifica si el parámetro p o d están activos, estos parámetros indican si el grafo será ponderado o dirigidos respectivamente. Si ambos están activos será un grafo ponderado y dirigido a la vez.

```
k = 3
self.c[i] = self.c[i] +1
self.c[j] = self.c[j] +1
self.A.insert(t,(self.x[i], self.y[i], self.x[j], self.y[j]))
t = t +1
```

Si grafo será ponderado entonces se genera un peso aleatorio para esa arista, será entero entre 1 y 10. Como buscamos que se visualice el peso a la hora de generar el grafo se utilizo la función que nos calcula el punto medio entre dos puntos. Ya habiendo calculado ese punto medio se guarda de manera que a la hora de visualizar el grafo se coloque en esa posición el peso de la arista. Mientras más peso tenga la arista más gruesa será.

```
if p:
    self.P.insert(t, math.ceil(random()*10))
    self.aux.insert(t, medio((self.x[i], self.y[i]),
    (self.x[j], self.y[j])))
```

Si el grafo es dirigido entonces se decide si la arista irá del nodo i al nodo j o si la dirección será del nodo j al nodo i. Para eso se escogió al azar entre el cero y el uno, de manera que si se escogía el

uno la arista dirigida iría del nodo i al nodo j, y si se escogía el cero la arista iría del nodo j al nodo i.

```
if d:
    self.D.insert(t,choice([0,1]))
```

Ya con todas las conexiones hechas y sabiendo si el grafo será simple, ponderado, dirigido o ambos, se procedió a hacer todos los preparativos para poderlo visualizar. De manera similar a la primera tarea, se guardaron las coordenadas de los nodos, así como el tamaño tendrá cada nodo y valor del color que le corresponde, luego de eso se verifica el tipo de grafo que será, para así hacer los preparativos adecuados para el grafo en particular. Los grafos tienen especificaciones comunes las cuales se encuentran a la hora de crear algunos de los grafos.

```
with open("nodos.dat", "w") as salida:
            for v in range(self.n):
                x = self.x[v]
                y = self.y[v]
                r = self.r[v]
                c = self.c[v]
with open("tarea2.plot", "w") as archivo:
                print("set term pdf", file = archivo)
                print("set key off", file = archivo)
                print("set output 'grafo.pdf'", file = archivo)
                print("set xrange [-0.1:1.1]", file = archivo)
                print("set yrange [-0.1:1.1]", file = archivo)
                print("set size square", file = archivo)
                print("show arrow", file = archivo)
                print("plot 'nodos.dat' using 1:2:3:4 with
                points pt 7 ps var lc palette frag var",
                file = archivo)
                print("quit()", file = archivo)
```

Si el grafo es simple, al igual que en la tarea anterior, simplemente se colocan los nodos con si correspondiente tamaño basado en su radio y un color según la cantidad de conexiones que tiene.

```
for i in range(len(self.A)):
    (x1, y1, x2, y2) = self.A[i]
    print("set arrow {:d} from {:f}, {:f} to {:f},
    {:f} lw 1 nohead ".format(num, x1, y1, x2, y2), file = archivo)
    num += 1
```

Si el grafo es ponderado, mantiene las mismas especificaciones que el grafo simple, con el cambio de que las aristas tienen un grosor adecuado a su peso y que se coloca el peso de la arista a la mitad de esta.

```
for i in range(len(self.A)):
    (x1, y1, x2, y2) = self.A[i]
    w = self.P[i]
    wi = w*0.3
    (xw, yw) = self.aux[i]
    print("set arrow {:d} from {:f}, {:f} to {:f}, {:f} lw
    {:f} nohead ".format(num, x1, y1, x2, y2, wi), file = archivo)
    print("set label '{:d}' at {:f},{:f}".format(w, xw, yw),
    file = archivo)
    num += 1
```

De forma similar para un grafo dirigido, las especificaciones se mantienen, pero ahora las aristas tendrán una dirección dependiendo de la condición antes mencionada. Las aristas para este grafo tienen un grosor constante.

```
for i in range(len(self.A)):
    (x1, y1, x2, y2) = self.A[i]
    if self.D[i] is 0:
        print("set arrow {:d} from {:f}, {:f} to {:f}, {:f}
        lw 1 backhead filled size 0.1,9"
        .format(num, x1, y1, x2, y2), file = archivo)
                        num += 1
    else:
        print("set arrow {:d} from {:f}, {:f} to {:f}, {:f}
        lw 1 head filled size 0.1,9 "
        .format(num, x1, y1, x2, y2), file = archivo)
    num += 1
```

El grafo dirigido y ponderado recoge las especificaciones clave de cada caso a la vez que conserva aquellas especificaciones comunes en cada grafo. Esto es, las aristas tienen una dirección dependiente de la condición ya mencionada, tienen un grosor dependiendo de su peso, además de que a la mitad de la arista se puede observar su peso.

```
for i in range(len(self.A)):
    (x1, y1, x2, y2) = self.A[i]
    w = self.P[i]
    wi = w*0.3
    (xw, yw) = self.aux[i]
    if self.D[i] is 0:
        print("set arrow {:d} from {:f}, {:f} to {:f}, {:f}
        lw {:f} backhead filled size 0.1,9"
        .format(num, x1, y1, x2, y2, wi), file = archivo)
        print("set label '{:d}' at {:f},{:f}"
        .format(w, xw, yw), file = archivo)
        num += 1
    else:
        print("set arrow {:d} from {:f}, {:f} to {:f}, {:f}
        lw {:f} head filled size 0.1,9 "
        .format(num, x1, y1, x2, y2, wi), file = archivo)
        print("set label '{:d}' at {:f},{:f}"
        .format(w, xw, yw), file = archivo)
        num += 1
```

Algunos de los grafos que fueron creados se muestran a continuación.

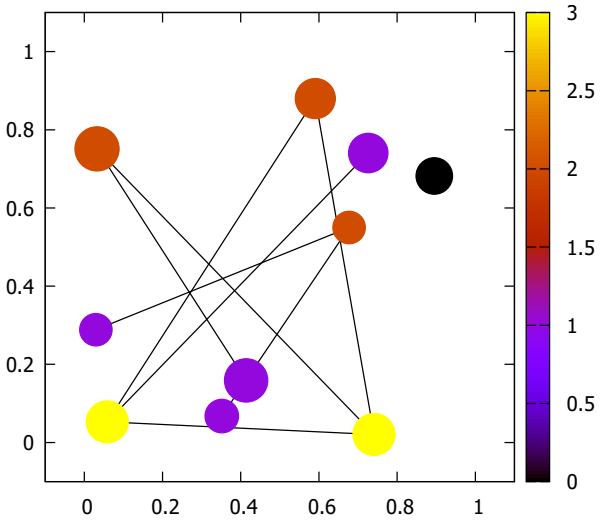

Figura 1: Grafo simple.

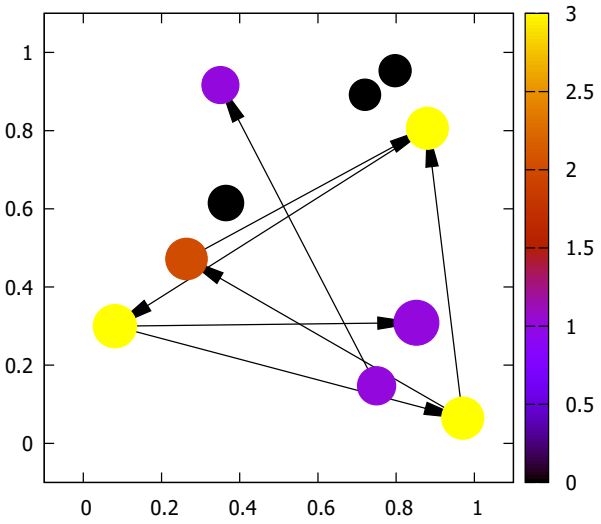

Figura 2: Grafo dirigido.

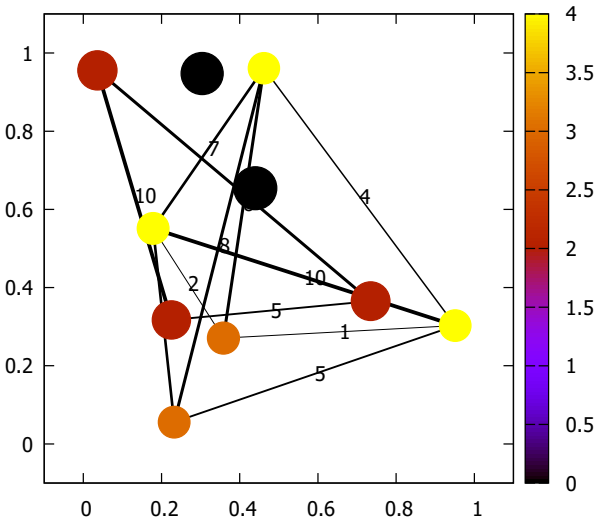

Figura 3: Grafo ponderado.

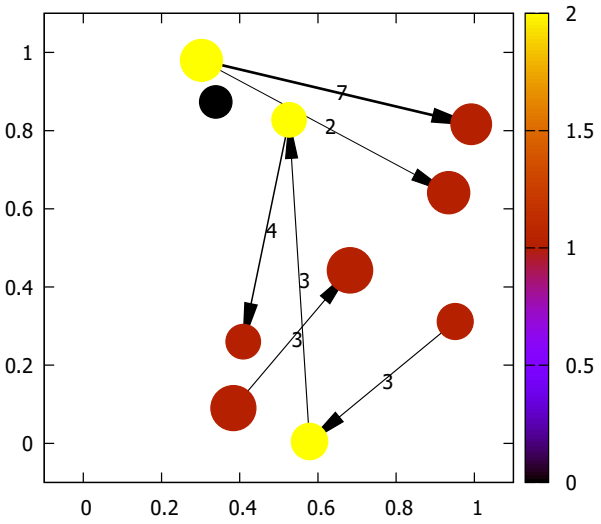

Figura 4: Grafo dirigido y ponderado.

3. Conclusiones

Al mejorar el código anterior se puede representar de mejor manera aquellos problemas donde es necesario medir o restringir lo que esta el flujo que puede haber en el grado, ademas de ya considerar una dirección en específico y el hecho de que podamos hacer un grafo que sea tanto dirigido como ponderado aumenta la cantidad de problemas que se pueden representar o resolver utilizando estas herramientas.