Tarea 2
Optimización de flujo en redes

Beatriz Alejandra García Ramos

5 de marzo de 2018

1. Introducción

En esta práctica se tiene la creación de un bosque en el que se puede decidir si se tienen direcciones y ponderaciones sobre las aristas. También en ésta se tiene que el bosque se crea a partir de la estructura de una clase en el programa `python` [1] con distintas características que deben ser señaladas en el programa `gnuplot` [2].

El diseñar el bosque a partir de una clase en `python` da la facilidad al usuario de crear tantos distintos grafos como quiera con el simple hecho de modificar los parámetros que se necesitan ingresar en el código que aportan las características del bosque, si se desea hacer dirigido o no, con ponderaciones o no y además si se desea cambiar la cantidad de nodos que tendrá el bosque.

2. Actualización del grafo en `python`

En la práctica anterior del curso de optimización de flujo en redes [3] se realiza un bosque en el cual se tienen ciertos nodos conectados entre sí por medio de aristas dependientes de un criterio, sin embargo el problema que se tiene en esa práctica es que el usuario no tiene la posibilidad de que se realicen bosques dirigidos o con ponderaciones ya que los criterios que se toman son únicos en el código realizado, lo cual ocasiona un problema si se requiere que alguna situación se visualice por medio de un grafo si éste necesita ser con dirección o ponderaciones como normalmente se utilizan en el área de flujo en redes.

Además se tiene en el código una gran cantidad de líneas que posiblemente al usuario no le interese conocer del todo, ya que el usuario se interesa en obtener el bosque final, es por eso que lo que se requiere es tener principalmente qué cantidad de nodos se requieren para el bosque y si éste debe ser dirigido, ponderado o ambos.

Es por esta razón que en esta práctica se tiene una clase que permite que el bosque se cree dependiente de los parámetros de la cantidad de nodos, direcciones y ponderaciones que el usuario indica en el llamado a las funciones definidas en la clase. Para que el bosque comience a crearse se necesita primeramente declarar los vectores, conjuntos y variables que se van a requerir en las demás funciones de la clase como se muestra a continuación.

```
class Grafo:

    def __init__(self):
        self.nodos = []
        self.aristas= []
        self.dis = []
        self.dismin = []
        self.peso = []
        self.vector = []
```

Para poder crear los nodos y conectarlos entre ellos se realizan las funciones `agrega`, `distancia` y `conecta` donde `agrega` define la coordenada de los nodos y además los va guardando en un archivo

.dat para que después se puedan graficar en gnuplot, los nodos se van tomando de manera aleatoria y una vez que se tienen todos en el vector nodos se calculan las distancias que existen entre ellos con la función distancia.

```
def agrega(self, n):
       with open("DirPesoNodos.dat", "w") as crear:
           for t in range(n):
               x=random()
               y=random()
               self.nodos.append((x,y))
               print (x,y,file=crear)

def distancia(self):
    for (x1,y1) in self.nodos:
        for (x2,y2) in self.nodos:
            d=sqrt(((y2-y1)**2)+((x2-x1)**2))
            if d==0:
                self.dis.append(20)
            else:
                if d in self.dis:
                    self.dis.append(20)
                else:
                    self.dis.append(d)
```

Una vez calculada la distancia ésta se utiliza en la función conecta la cual toma la menor distancia entre dos nodos que se encuentra en el vector dis, verifica si ésta se encuentra en el vector dismin y si no es así entonces conecta esos nodos entre sí, pero para ello se requiere saber cuál es la distancia mínima de todas las distancias calculadas, lo cual podría ser un conflicto si la distancia que se calcula entre un nodo y sí mismo es cero dado que siempre tomaría esas distancias y lo que se requiere es que cada nodo tenga al menos una conexión con otro, es por eso que la distancia entre un nodo y sí mismo en la función distancia es veinte cuando esto sucede.

```
def conecta(self):
    for (x1,y1) in self.nodos:
        self.dismin=[]
        for(x2,y2) in self.nodos:
            dm=sqrt(((y2-y1)**2)+((x2-x1)**2))
            if dm==0:
                  self.dismin.append(20)
            else:
                if dm in self.dismin:
                    self.dismin.append(20)
                else:
                    self.dismin.append(dm)
        if min(self.dismin) in self.dis:
            a1=self.nodos[self.dismin.index(min(self.dismin))][0]
            b1=self.nodos[self.dismin.index(min(self.dismin))][1]
            self.aristas.append((x1,y1,a1,b1))
            self.peso.append(ceil(random()*10))
            self.vector.append(punto((x1,y1),(a1,b1)))
            self.dis.remove(min(self.dismin))
```

Una vez que se realiza la conexión de un nodo con otro se toma de manera aleatoria una ponderación para esa arista que toma valores enteros entre un intervalo de 1 y 10 incluyéndolos, además se calcula la posición en que esta ponderación será colocada en el gráfico y se requiere para ello una función definida fuera de la clase que calcula el valor de las coordenadas de ese punto.

```
def punto(x1,y1):
    return(((x1[0] + y1[0])/2)-.005, ((x1[1] + y1[1])/2)+.005)
```

3. Resultados

Para poder graficar los datos obtenidos en `gnuplot` se requieren ciertas instrucciones que indican si el bosque será ponderado o con direcciones.

Para que un bosque sea dirigido o no se requiere de la instrucción `head` y `nohead` respectivamente, para lo cual tendremos que decidir, si la variable `di` que proporciona el usuario es cero entonces el bosque tomará la instrucción de `nohead`, si es uno tomará la instrucción de `head` que nos dará direcciones en éste.

```
if di is 1:
   print("set arrow {:d} from {:f}, {:f} to {:f}, {:f} head filled lw 1".format(num+1,x1,y1,x2
       ,y2),file=archivo)
else:
   print("set arrow {:d} from {:f}, {:f} to {:f}, {:f} nohead filled lw 1".format(num+1,x1,y1,
       x2,y2),file=archivo)
```

Entonces para indicar que el bosque debe tener dirección la variable `di` es igual a uno, lo cual nos da como resultado grafos como los de la figura 1 en donde las direcciones se toman dependiendo de qué nodo es el que se está conectando con otro.

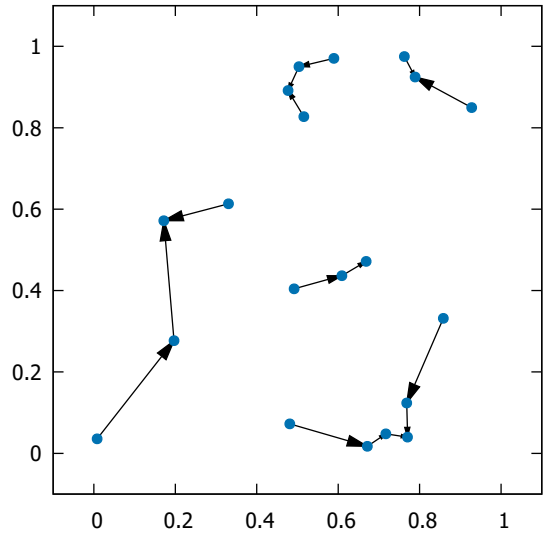
Para que un bosque sea ponderado entonces se deben tomar los datos de los vectores `peso` y `vector` que son los que indican en qué posición se pondrá la ponderación que le corresponde a cada arista. Así para la primer arista se tomaría en cuenta que $\texttt{p} = \texttt{self.peso}[0]$ y $(\texttt{kp}, \texttt{rp}) = \texttt{self.vector}[0]$ que son los datos de la primer posición de cada vector. De esta manera cada arista tendría el valor que le corresponde dado en la función `conecta`.

```
if pesos is 1:
   p = self.peso[num]
   (kp, rp) = self.vector[num]
   print("set label font ',11' '{:d}' at {:f},{:f} tc rgb 'brown'".format(p, kp, rp),file =
       archivo)
```

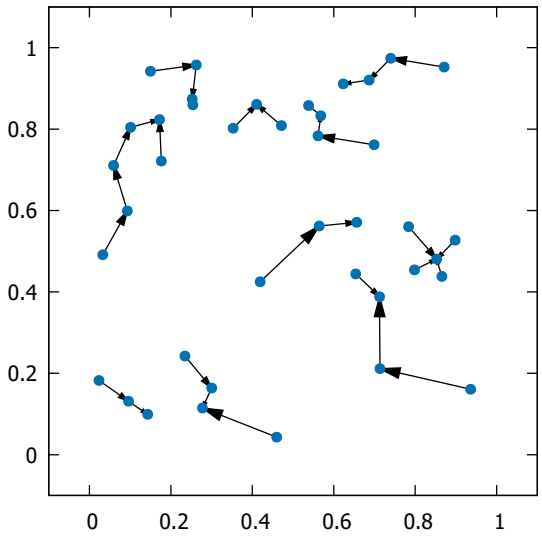
Indicando que la variable `pesos` es igual a uno tendremos un bosque ponderado como los que se encuentran en la figura 2 que varían en nodos y tiene ponderaciones determinadas en un intervalo entre uno y diez tomando los enteros como se puede observar en los valores indicados en cada arista, donde `i` es la cantidad de nodos del bosque.

De la misma manera se tendrá que cuando ambas variables sean iguales a uno el bosque será dirigido y ponderado de manera aleatoria como se observa en la figura 3.

```
i=40
G = Grafo()
G.agrega(i)
G.distancia()
G.conecta()
G.graficar(di=1, pesos=1)
```

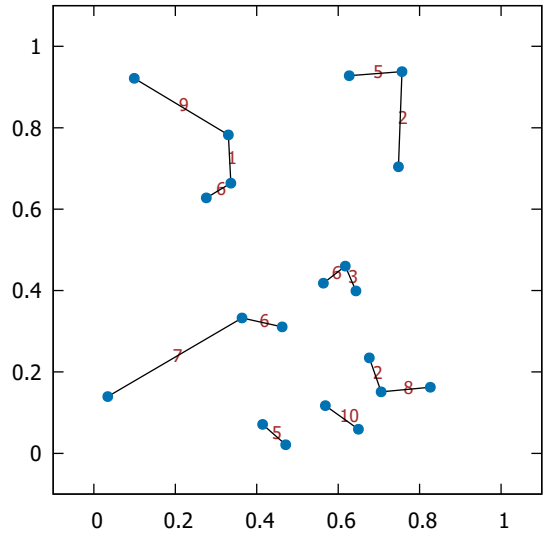


(a) 20 nodos.

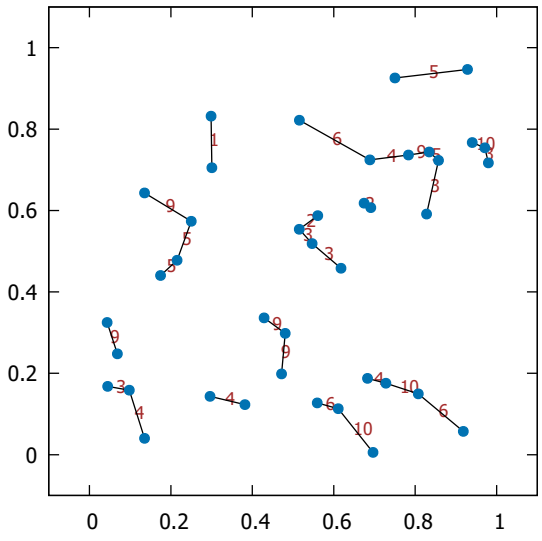


(b) 40 nodos.

Figura 1: Grafos de bosque dirigidos.

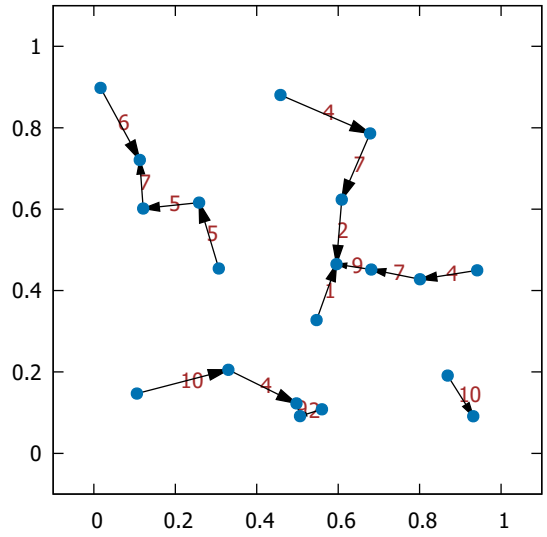


(a) 20 nodos.

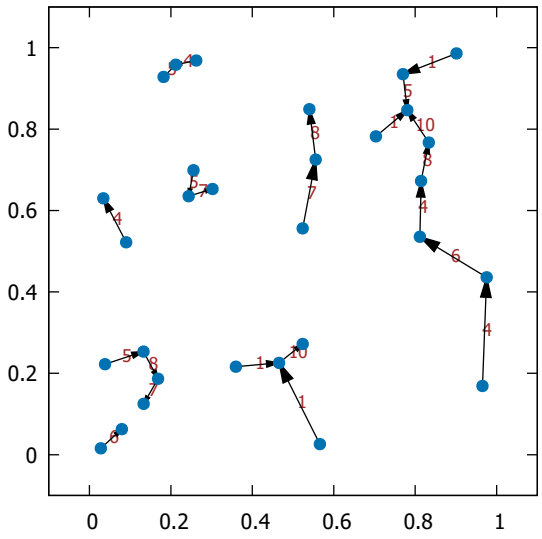


(b) 40 nodos.

Figura 2: Grafos de bosque con ponderaciones.



(a) 20 nodos.



(b) 30 nodos.

Figura 3: Grafos de bosque dirigidos y con ponderaciones.

Referencias

[1] Python Software Foundation [US]. (2001-2018). https://www.python.org/downloads/

[2] Gnuplot. http://www.gnuplot.info/

[3] Beatriz García. Tarea 1. Optimización de flujo en redes. https://github.com/BeatrizGarciaR/FlujoEnRedes/blob/master/Tarea1/Tarea1.pdf, 2018.