Tarea 2: Grafos simples, ponderados y dirigidos

María Gabriela Sandoval Esquivel

Febrero 2018

1 Introducción

El objetivo de esta tarea es mejorar el programa que tenemos para trazar grafos en Gnuplot [1] incorporando el uso de clases para agilizar el proceso y añadiendo características de estilo a las aristas. Las clases se definen en nuestro programa de Python [2] para poder crear objetos con los atributos deseados para los grafos. Su uso agiliza el proceso para generar archivos con instrucciones para graficar diferentes tipos de grafos en Gnuplot.

2 Estilos aristas

Los grafos simples, ponderados y dirigidos se distinguen por los atributos que se les dan a sus aristas. En un grafo simple las aristas representan conexiones reflexivas entre dos nodos y no se considera ninguna otra característica que distinga a una arista de otra. Visualmente las aristas son líneas simples que conectan a dos nodos y todas las aristas de un grafo simple son iguales. La instrucción para trazar las aristas de un grafo simple en Gnuplot es la siguiente:

```
set arrow <arrow_number> from <x1,y1> to <x2,y2> nohead
```

En un grafo ponderado a las aristas se les asignan diferentes pesos y visualmente esto se puede representar cambiando el grosor de la arista según su peso. La instrucción en Gnuplot sería la siguiente:

```
set arrow <arrow_number> from <x1,y1> to <x2,y2> nohead lw <arrow_width>
```

Finalmente, un grafo dirigido indica que hay diferencia entre los nodos que conectan un grafo. Por ejemplo, un nodo puede ser el nodo de salida mientras que el segundo el de llegada. De esta manera se distingue entre la arista que va del nodo 'a' al nodo 'b' y la que va del nodo 'b' al 'a'. Visualmente las aristas de grafos dirigidos se representan con flechas y para trazarlas en Gnuplot se usa la siguiente instrucción:

```
set arrow <arrow_number from <x1,y1> to <x2,y2> head
```

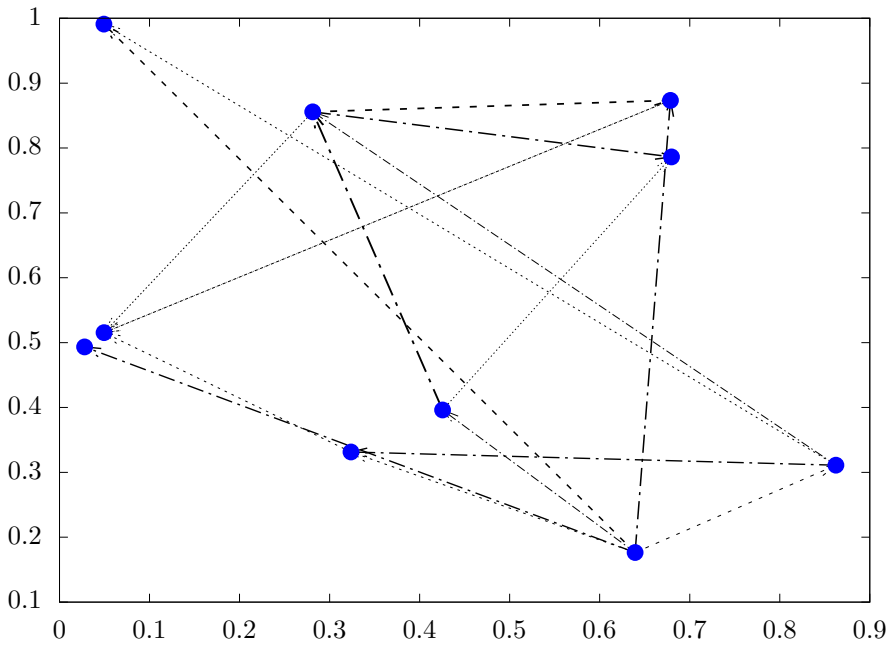

Figure 1: Ejemplos de estilos de aristas

Se puede indicar también si deseamos que la fleche tenga diferentes estilos con las siguientes instrucciones:

```
set arrow <arrow_number from <x1,y1> to <x2,y2> head filled\\
set arrow <arrow_number from <x1,y1> to <x2,y2> head nofilled\\
set arrow <arrow_number from <x1,y1> to <x2,y2> head empty
```

3 Definir Clases

En programación orientada a objetos se hace uso de clases para definir tipos de objetos. Al crear una clase se pueden definir los campos (o variables) y los métodos que estarán relacionados a los objetos que pertenezcan a esa clase [3] Los campos (fields en inglés) son atributos de los objetos de esa clase. Al crear la clase de Grafos, por ejemplo, se puede definir el campo de los nodos y el campo de las aristas. Los métodos son funciones que se aplican a los objetos de esa clase. En este caso tendremos funciones para agregar nodos y aristas al grafo, entre otras.

3.1 Campos

Como ya vimos los campos básicos para la clase de grafos son los nodos y las aristas. En el programa definimos a los nodos como un conjunto y a las aristas como diccionarios que relacionan a los elementos del conjunto de nodos. Los nodos los describimos con un nombre (variable tipo string) y se les atribuye un par de coordenadas, y variables numéricas para asignar su tamaño y su color en caso de que sea necesario en el grafo. Las aristas por su parte se describen como una pareja de nodos, se les atribuye una variable de peso (variable numérica), se indica si son dirigidas (variable booleana) y el estilo de la línea punteada (variable entera). Otro campo que se define para los grafos es el de vecinos que es un diccionario que relaciona a cada nodo con el conjunto formado por los nodos que están conectados con él por una arista.

3.2 Métodos

Los métodos de una clase son funciones que se definen para sus objetos y atributos. A continuación, está la descripción de las funciones que se han definido hasta ahora para la clase de grafos.

3.2.1 Agrega

Esta función agrega elementos al conjunto de nodos indicando las características de posición, tamaño y color. Si no se desea especificar alguna de estas características se asignan valores por default para un nodo simple.

```
def agrega(self, v,tamano = 1, posicion = (0,0), color = 0):
    self.nodos.add(v)
    self.tamano[v] = tamano
    self.posicion[v] = posicion
    self.color[v] = color
    if not v in self.vecinos:
        self.vecinos[v] = set()
```

3.2.2 Conecta

Con esta función se indica que existe una arista entre dos nodos y las características específicas de esa arista. Se puede hacer uso de esta función con nodos que no hayan sido definidos previamente, en este caso la función conecta llama a la función agrega con valores default para las características de los nodos.

```
def conecta(self, v, u, peso=1, dirigido = False, tipo = 1):
    if not v in self.nodos:
        self.agrega(v)
    if not u in self.nodos:
        self.agrega(u)
    self.vecinos[v].add(u)
    self.vecinos[u].add(v)
    if dirigido:
        self.aristas[(u, v)] = (peso, tipo, dirigido) # en ambos
    sentidos
```

```
    else:
        self.aristas[(v, u)] = self.aristas[(u, v)] = (peso, tipo,
    dirigido)
```

3.2.3 Complemento

Esta función se usa para definir otro objeto de la clase Grafo. El grafo complemento contiene el mismo conjunto de nodos que el original, pero sus aristas corresponden a las que no existen en el grafo original.

```
    def complemento(self):
        comp= Grafo()
        for v in self.nodos:
            for w in self.nodos:
                if v != w and (v, w) not in self.aristas:
                    comp.conecta(v, w, 1)
        return comp
```

3.2.4 Aleatorio

Con esta función se crea un grafo especificando un número de nodos deseado y una probabilidad de conexión. De este modo se creará una arista entre todo par de nodos de acuerdo a esta probabilidad.

```
def aleatorio(self, n=10, prob = 0.5, dirigido = False,
    rand_style_aristas = False):
    from random import random
    for i in range(n):
        tag_nodo = str(i)
        x = random()
        y = random()
        size = random()
        color = random()
        self.agrega(tag_nodo, size, (x,y),color)
    for i in range(n - 1):
        for j in range(i + 1, n):
            if random() < prob:
                if rand_style_aristas:
                    atipo = int(10*random()) % 5
                    peso = 3*random()
                else:
                    atipo = 1
                    peso = 1
                self.conecta(str(i),str(j),peso, dirigido,atipo)
```

3.2.5 Lee

Con esta función se crea un grafo de acuerdo con datos de un archivo de texto con un formato específico. La primera línea de ese archivo debe de indicar el número de nodos y los renglones siguientes deben de indicar las características de esos nodos. La primera columna corresponde al índice del nodo, las siguientes dos las coordenadas de posición del nodo seguidas de la columna que indica el

tamaño y finalmente el color. Después de la información de los nodos sigue un renglón que indica el número de aristas y después los renglones que indican las parejas de índices de los nodos que forman a cada arista. Este formato es el que tienen las instancias que uso para mi proyecto de tesis y con el que puedo visualizar este tipo de datos.

```
def leer(self, filename = "2DU80-05-2.dat"):
    with open(filename, 'r') as archivo:
        nn = int(archivo.readline())
        for i in range(nn):
            stringLine = archivo.readline()
            splitLine = stringLine.split(" ")
            valueList = [float(e) for e in splitLine]
            index, x, y, a, b, c = valueList
            index = int(index)
            idx = str(index)
            #print(idx)
            a = pow((a - 590),3)
            self.agrega(idx, a, (x,y), 1)
        neigh = int(archivo.readline())
        for i in range(neigh):
            stringLine = archivo.readline()
            splitLine = stringLine.split(" ")
            neigh1, neigh2 = splitLine
            neigh2, basura = neigh2.split("\n")
            self.conecta(neigh1,neigh2,1)
    archivo.close()
```

3.2.6 Gnuplot

Finalmente está la función que se nombró gnuplot que sirve para crear un archivo con las instrucciones necesarias para que se trace el grafo con las características deseadas. A esta función se le puede indicar el nombre del archivo y si se desea que se consideren los colores y tamaños de los nodos para la visualización. En esencia esta función contiene la mayoría de las instrucciones de la versión anterior del programa para trazar grafos.

```
def gnuplot(self, gcolor = True, gtamano = True, name = "grafo_a"):
    n = len(self.nodos)
    with open("nodos.dat",'w') as archivo_nodos:
        for nodo in self.nodos:
            #nodo = str(i)
            (x,y) = self.posicion[nodo]
            size = self.tamano[nodo]
            color = self.color[nodo]
            print(x, y,size,color, file = archivo_nodos)
    filename = name + ".plt"
    imagename = " '"+ name + ".tex' "
    arrow_idx = 1
    archivo_nodos.close()
    with open(filename, 'w') as archivo:
        print("set term epslatex", file = archivo)
        print("set output"+imagename, file = archivo)
        print("set pointsize 2", file = archivo)
        print("unset arrow", file = archivo)
```

```
        print("unset colorbox", file = archivo)
        for (ii,jj) in self.aristas:
            (x1, y1) = self.posicion[ii]
            (x2, y2) = self.posicion[jj]
            (apeso, atipo, adirected) = self.aristas[(ii,jj)]
            head = "nohead"
            if adirected:
                head = "head"
            print("set arrow", arrow_idx, "from", x1, "," ,y1," to
", x2, ",", y2, head, " lw ",apeso, " dashtype ", atipo, file =
 archivo)
            arrow_idx += 1
        #print("set style fill transparent solid 0.7", file =
archivo)
        print("set style fill solid", file = archivo)
        print("set palette defined (0 'blue', 3 'green', 6 'yellow
', 10 'red') ", file = archivo)
        if gcolor and gtamano:
            print("color y tama o!!")
            print("plot 'nodos.dat' using 1:2:(sqrt(3)/30):4 with
circles palette notitle", file = archivo)
        elif gcolor:
            print("sin tama o!!")
            print("plot 'nodos.dat' using 1:2:3 with points pt 7
palette notitle", file = archivo)
        elif gtamano:
            print("sin color!!")
            print("plot 'nodos.dat' using 1:2:(sqrt(3)/30) with
circles notitle", file = archivo)
        else:
            print("sin color sin tama o!!")
            print("plot 'nodos.dat' using 1:2 with points pt 7 lc
rgb 'blue' notitle", file = archivo)

        archivo.close()
```

4 Conclusión

Definir una clase para definir grafos como objetos tiene varias ventajas tanto para la manipulación de estos datos en Python como para crear los archivos de instrucción para gnuplot de manera más eficiente. La estructura que se definió aquí para los objetos de la clase grafo es útil para manejar distintos atributos y sus relaciones dentro de Python. Por ejemplo, se puede acceder fácilmente al conjunto de vecinos de un nodo en específico o al conjunto de aristas de un grafo solamente. Esto resultó útil para indicar la instrucción del trazo de aristas y puede ser útil en varias otras aplicaciones. Por otro lado, esta estructura y sus funciones permiten que fácilmente se puedan definir o leer varios grafos con sus características específicas y crear las instrucciones para trazar grafos con diferentes formatos en pocas líneas de código.

References

[1]

[2] Guido Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[3] Swaroop C. H. *A Byte of Python*. CreateSpace Independent Publishing Platform, USA, 2015.