Arbol recubridor minimo

Jorge Steven Reyes - 1667619
Luis Eduardo Henao - 1667483
Jorge Augusto Estacio - 1667409
Matematicas discretas II, Escuela de ingenieria de sistemas y computacion
Universidad del Valle
Tulua

10/12/2017

Resumen

Se pretende explicar el funcionamiento del programa realizado en Python para dibujar grafos con sus arboles recubridores minimos gracias a un algoritmo muy famoso llamado 'Algoritmo de Kruskal' e implemenetando las librerias NetworkX, Matplotlib, Tkinter y Numpy.

1 Como se integraron las librerias

1.1 Networkx

NetworkX es un paquete de Python para la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas.
Nos permitio convertir el grafo representado en su matriz de adyacencia a la propia estructura de grafos de la libreria la cual es muy intuitiva.
Aqui un ejemplo creando un grafo sencillo:

```
import networkx as nx
G=nx.Graph()
G.add_node(a)
G.add_node(b)
G.add_edge(a,b)
```

De esta forma se dibuja el grafo de una forma muy basica:

```
nx.draw_networkx_nodes(G,pos)
nx.draw_networkx_edges(G,pos,width=2)
```

Asi se implemento Networkx
1) Creamos el grafo G y añadimos la informacion necesaria (vertices, aristas y key) con base en la informacion del array MA, por defecto el key sera 0.
2) Utilizando $pos = nx.circular_layout(G)$ le damos la forma al grafo, en este caso sera circular. (Probando las distintas formas concluimos que esta es la mas adecuada para mostrar adecuadamente el grafo).
3) Creamos el grafo G1 y añadimos la informacion necesaria (vertices y aristas) con base en la informacion del array MAA que contiene la matriz resultante del algoritmo de kruskal.
4) Creamos la lista LG1 que contiene la informacion de las aristas del grafo G1 5) Utilizando un ciclo for se sobreescriben las aristas de G que estan contenidas en G1 pero se les cambia el key a 1.
6) Creamos una lista llamada "caminoA" con las aristas de G que tengan key=1.
7) Para la venatana 1 donde se encuentra el grafo normal dibujamos el grafo completo utilizando las funciones 'nx.draw' dibujando de esta forma los vertices, aristas, labels de los pesos y los nodos.
8) Para la ventana 2 donde se muestra en arbol recubridor minimo dibujamos dibujamos de la misma manera que la ventana 1, solo que se todas las aristas se dibujan con un alpha (hace que se vean mas claras) y luego se dibujan las aristas del "caminoA"(Que son las que importan) encima de las demas, todo este dibuja se hace en un diferente canvas que el primer dibujo de la ventana 1.

1.2 Numpy

La libreria Numpy tiene muchas funciones pero en este caso solo se implemento para crear dos listas de una forma sencilla (Estas listas son utilizadas por el algoritmo de kruskal y MAA es el resultado de dicho algoritmo).

```
PE = np.arange(dim) # PE lista de vertices
MAA = np.zeros((dim,dim)) # MAA matriz con ceros
```

1.3 Tkinter

Gracias a Tkinter creamos las dos ventanas, *minist* y *win*. estas seran los espacion en donde se añadiran los canvas de los dibujos.
Principalmente añadimos los items generados por matplotlib y creamos un boton "Mostrar arbol recubridor minimo" el cual su comando es la funcion *mostrar* la cual muestra la ventana *win* con todo su contenido.
Para mostrar la ventana, al final de codigo se utiliza $Tk.mainloop()$

1.4 Matplotlib

Matplotlib es una libreria de trazado 2D de Python que produce figuras de calidad de publicación en una variedad de formatos impresos y entornos interactivos en todas las plataformas. La utilizamos primero para crear las figuras en donde estaran los plots de los grafos generados por networkx.

```
fi = Figure(figsize=(12,7), dpi=100 )
a = fi.add_subplot(111)
nx.draw_networkx_nodes(G,pos,ax=a)
```

Podemos notar que todos los dibujos que se quieran añadir a la figura deben contener $ax = a$ donde a es el subplot de la figura, de esta forma para la ventana *win* (Donde se debe mostrar el arbol) creamos otra figura y se añade de nuevo el dibujo del grafo pero en otra figura. Tambien añadimos toolbars en las ventanas en donde matplotlib nos permite modificar, navegar, ampliar y hasta guardar al dibujo como png.

1.5 sys

Sys se usa para la entrada estandar *python proyectoMST.py < archivo.txt*

```
f = sys.stdin
data = f.read().strip()
f.close()
MI = [[int(num) for num in line.strip().split()] for line in data.split('\n')]
```

De esta forma todo el contenido del archivo txt queda en *data* y luego llenamos la lista MI con esta informacion, de la cual se sacan la variable *dim* y la matriz MA que es la matriz de adyacencia.

2 Algoritmo de Kruskal

El algoritmo de Kruskal es un algoritmo de la teoria de grafos para encontrar un arbol recubridor minimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un arbol, incluyen todos los vertices y donde el valor de la suma de todas las aristas del arbol es el minimo.

Para explicar mejor como se implemento en el proyecto a continuacion el codigo del algoritmo explicado paso a paso

```
nodos = dim #cantidad de nodos del grafo y del arbol
A=0 #nodo 0
B=0 #nodo 0
cantidadDistancias = 1 #distancias que pueden tener el arbol (como minimo debe
    existir al menos 1 distancia para ser arbol)

#Se crea PE y MAA
PE = np.arange(dim) # PE lista de vertices
MAA = np.zeros((dim,dim)) # MAA matriz con ceros

while cantidadDistancias != nodos: # mientras la cantidad de distancias no sea
    igual a los nodos (para ser un arbol se debe cumplir eso)
    mini = 999 #cantidad minima de una arista o distancia, por defecto es 999
    actual = 0 #Indica el nodo en el que se esta posicionando actualmente
    for i in range(nodos): #se itera en la matriz de adyacencia recibida y se
        verifica
```

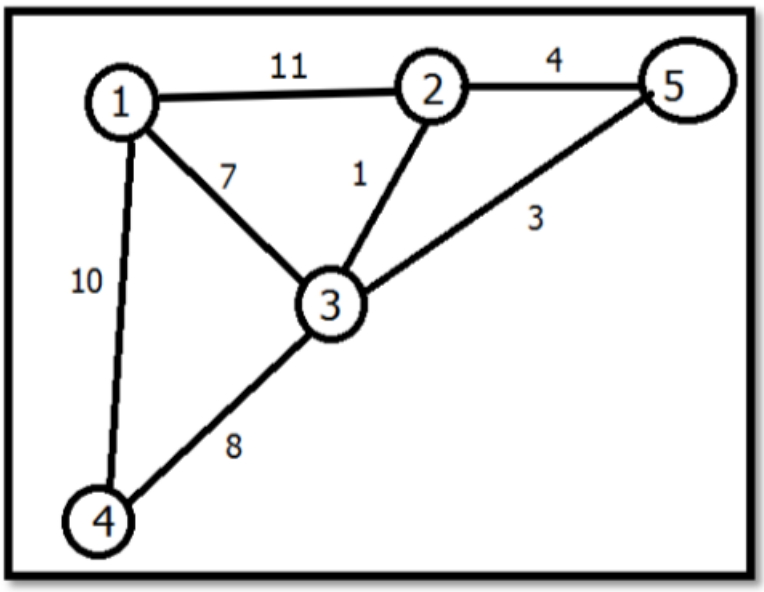
```
        for j in range (nodos):
            if mini > MA[i][j] and MA[i][j] != 0 and MA[i][j] != -1 and PE[i]
               != PE[j]: #si minimo es mayor a la distancia en la interseccion
                (i,j)(para saber si es un valor minimo)
                mini = MA[i][j]
                                                                    #Si
                    la distancia en la intersecion es diferente de 0 y -1 (
                   para saber si almenos existe una distancia entre los nodos
                   intersectados)
                A=i

                                        #y si el elemento de la lista en
                   posicion i es diferente a el de la lista en posicion j
                B=j

                                        #(si son iguales quiere decir que
                   la distancia  ya esta incluida en el arbol,si no es lo
                   contrario a lo anterior V:)
                               #ahora el minimo es la distancia minima
                                   encontrada, nodo A=i y nodo B= j

    if PE[A] != PE[B]: #Se pregunta por segunda vez si el elemento de la lista
         en A es diferente a el de la lista en B (A y B ademas de ser nodos
        indican una posicion)
        MAA[A][B]= mini #se coloca en la matriz del arbol recubridor el valor
            minimo en la interseccion (A,B) (representa la arista o distancia
            minima de A a B)
        MAA[B][A]= mini #Como los arboles son simetricos se hace la contra
            parte (representa la arista o distancia minima de B a A)
        actual = PE[B] #nodo actual pasa a ser el ultimo nodo
        PE[B] = PE[A]  #el ultimo nodo pasa a ser el nodo con el que conecta (
            tecnicamente esto representa la union de ese nodo con otro para
            formar o seguir formando un arbol, donde PE[A] se tendra en cuenta
            como raiz)
        for k in range (nodos): #Se itera en la lista para encontrar un nodo
            igual al actual (si encuentra una union entre un nodo cualquiera y
            el nodo actual
            if PE[k] == actual:

                                        #entonces la conexion entre ese nodo
               no sera con el actual, si no con la raiz).
                PE[k]=PE[A]
    cantidadDistancias = cantidadDistancias+1 #se aumenta la cantidad de
        distancias y se verifica si cumple la condicion del while
```

3 Prueba de escritorio

3.1 elementos

Para la siguiente prueba de escritorio en la que aplicaremos el algoritmo de Kruskal implementado en la aplicacion, usaremos el siguiente grafo conexo, no dirigido de 5 vertices.

Grafo A:



La representación del grafo A en una matriz de adyacencia es la siguiente:

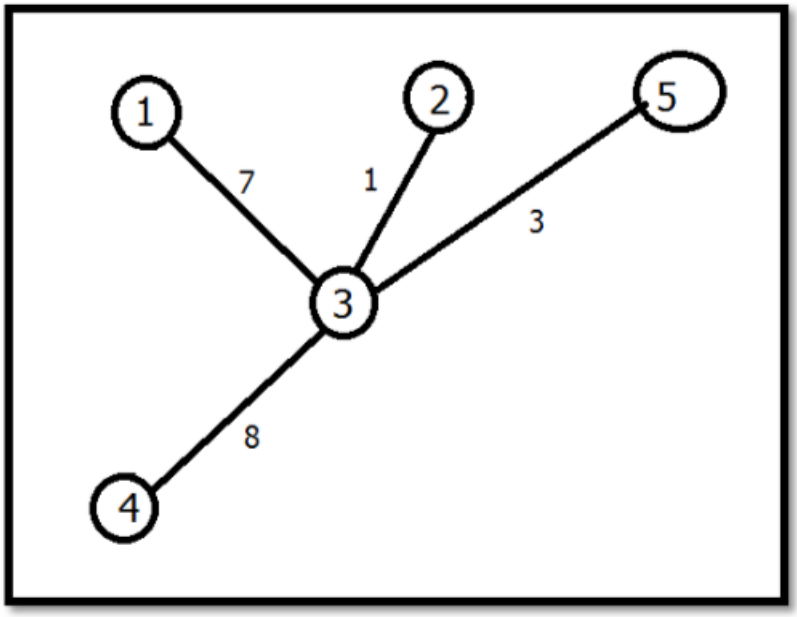
Matriz A

	1	2	3	4	5
1	0	11	7	10	0
2	11	0	1	0	4
3	7	1	0	8	3
4	10	0	8	0	0
5	0	4	3	0	0

Para este algoritmo, los pesos de los vértices que no conecten con otro vertice o con si mismos se representarán con 0.
La Matriz A es la que tomaremos como dato de entrada, luego de haber conocido la cantidad de vertices de este grafo (5).
Lo que nos debe retornar este algoritmo es el arbol recubridor mínimo de la matriz recibida, el arbol recubridor minimo sera representado como una matriz.
Por lo que, una vez tomada la Matriz A, y aplicado el algoritmo de Kruskal, tendremos la siguiente matriz:
Y su reprsentacion grafica seria:

MAA

	1	2	3	4	5
1	0	0	7	0	0
2	0	0	1	0	0
3	7	1	0	8	3
4	0	0	8	0	0
5	0	0	3	0	0



3.2 Aplicación del algoritmo.

dim = 5 (Numero de vertices del grafo y Nodos del arbol)
MA [5] [5] = Matriz A (Matriz que representa el grafo A)
Kruskal:
Nodos= 5
A=0 representa un nodo
B=0 representa un nodo que conecta con A
cantidadDistancias =1;
debe existir al menos 1 distancia en un arbol para que se le considere como tal

PE= 0, 1, 2 ,3 ,4 MAA [5] [5]=0; Inicialmente la matriz que representa el arbol, es una matriz de ceros
Mientras cantidadDistancias (1) sea diferente de Nodos (5) HACER:
    mini= 500 valor minimo por defecto

actual=0 nodo actual por defecto

Se itera para recorrer MA [5] [5] y pregunta en cada intersección [posFila][posColumna]:

Si (mini > MA [posFila] [posColumna], y MA [posFila] [posColumna] es diferente de 0 y -1, y PE [posFila] es diferente a [posColumna]) Hacer:

Para ahorrarnos todo el procedimiento usaremos el valor de MA [1][2]=1; debido a que es el primer valor minimo en la matriz.

mini= MA [1] [2] =1 valor minimo de la matriz
A= [posFila]=1 nodo
B= [posColumna]=2 nodo que conecta con el anterior
Fin si.
Fin iteracion
Si (PE [1]=1 es diferente a PE [2]=2) Hacer:
MAA [1] [2] = mini=1
MAA [2] [1] = mini=1
actual= PE[2] = 2
PE [2] = PE[1]= 1
Se itera desde k=0 hasta k=nodos veces para recorrer PE [] y se pregunta a cada elemento:
Si (PE[k] es igual a actual=2) Hacer:

Para ahorrarnos todas la iteraciones, analizaremos la lista y obtendremos que no existe un PE[K]=2, por lo ue no se hace nada.

PE [k] = PE[1]
Fin si.
Fin iteracion.
Fin si.
cantidadDistancias+1 = 2;
Fin mientras (cantidadDistancias (2) sea diferente de nodos (5)).

Y PE[] = 0, 1, 1, 3, 4

Despues de esta iteracion nos quedan faltando 3 iteraciones mas:

Mientras cantidadDistancias (2) sea diferente de Nodos (5) HACER:
mini= 500 valor minimo por defecto
actual=0 nodo actual por defecto

Se itera para recorrer MA [5] [5] y pregunta en cada interseccion [posFila][posColumna]:

Si (mini > MA [posFila] [posColumna], y MA [posFila] [posColumna] es diferente de 0

En la primera iteración la matriz del árbol recubridor mínimo (MAA) es igual a:

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	1	0	0
3	0	1	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Tener en cuenta que las posiciones de las matrices inician desde 0.

y -1, y PE [posFila] es diferente a [posColumna]) Hacer:

Para ahorrarnos todo el procedimiento usaremos el valor de MA[2][4]=3; debdo a que es el primer valor minimo en la matriz.

mini= MA [2] [4] =3 valor minimo de la matriz
A= [posFila]=2 nodo
B= [posColumna]=4 nodo que conecta con el anterior
Fin si.
Fin iteracion
Sí (PE [2]=1 es diferente a PE [4]=4) Hacer:
MAA [2] [4] = mini=3
MAA [4] [2] = mini=3
actual= PE[4] =4
PE [4] = PE[2]= 1
Se itera desde k=0 hasta k=nodos veces para recorrer PE [] y se pregunta a cada elemento:
Sí (PE[k] es igual a actual=4) Hacer:

Para ahorrarnos todas la iteraciones, analizaremos la lista y obtendremos que no existe un PE[K]=4, por tanto no se hace nada.

PE [k] = PE[2]
Fin sí.
Fin iteracion.
Fin si.
cantidadDistancias+1 = 3;
Fin mientras (cantidadDistancias (3) sea diferente de nodos (5)).

En la primera iteración la matriz del árbol recubridor mínimo (MAA) es igual a:

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	1	0	0
3	0	1	0	0	3
4	0	0	0	0	0
5	0	0	3	0	0

Tener en cuenta que las posiciones de las matrices inician desde 0

Y PE[] = 0, 1, 1, 3, 1

Despues de esta iteracion nos quedan faltando 2 iteraciones mas, pero en estas se realiza el mismo proceso:

Obtener un valor minimo del árbol min, junto su posicion en fila A y columna B. (A y B de igual forma son nodos que conectan entre ellos).
Introducir el valor minimo en la matriz de arbol recubridor minimo en su intersección correspondiente.
(Teniendo en cuenta las variables A, B y actual)Obtener un nodo actual= PE[B] , igualar el nodo PE[B] al nodo con el que conecta PE [A] y buscar en la lista PE un elemento que sea igual al nodo actual (PE[B]) y si lo encuentra igualar PE[posiciónElemento] al nodo PE[A].

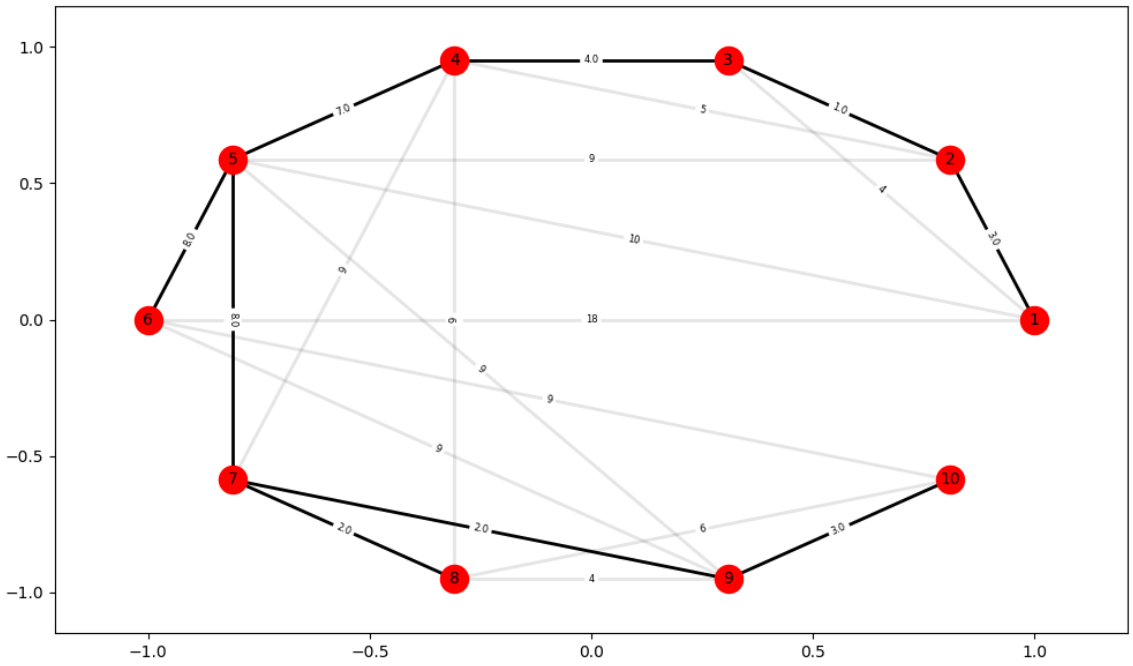
4 Pruebas

4.1 Prueba1

Grafo: 10 vertices, 21 aristas
Matriz entrante:

0	3	4	0	10	18	0	0	0	0
3	0	1	5	9	0	0	0	0	0
4	1	0	4	0	0	0	0	0	0
0	5	4	0	7	0	9	9	0	0
10	9	0	7	0	8	8	0	9	0
18	0	0	0	8	0	0	0	9	9
0	0	0	9	8	0	0	2	2	0
0	0	0	9	0	0	2	0	4	6
0	0	0	0	9	9	2	4	0	3
0	0	0	0	0	9	0	6	3	0

Matriz y grafica resultante:



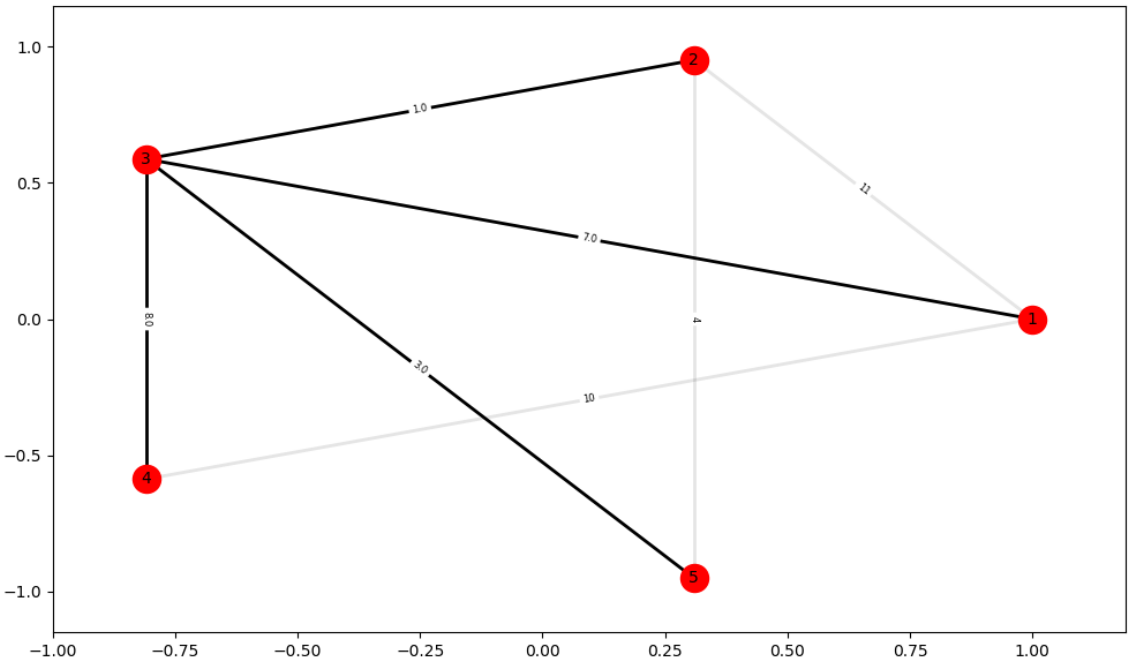
0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	4.0	0.0	7.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	7.0	0.0	8.0	8.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	8.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	8.0	0.0	0.0	2.0	2.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	3.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0

4.2 Prueba2

Grafo: 5 vertices, 7 aristas
Matriz entrante:

0	11	7	10	0
11	0	1	0	4
7	1	0	8	3
10	0	8	0	0
0	4	3	0	0

Matriz y grafica resultante:



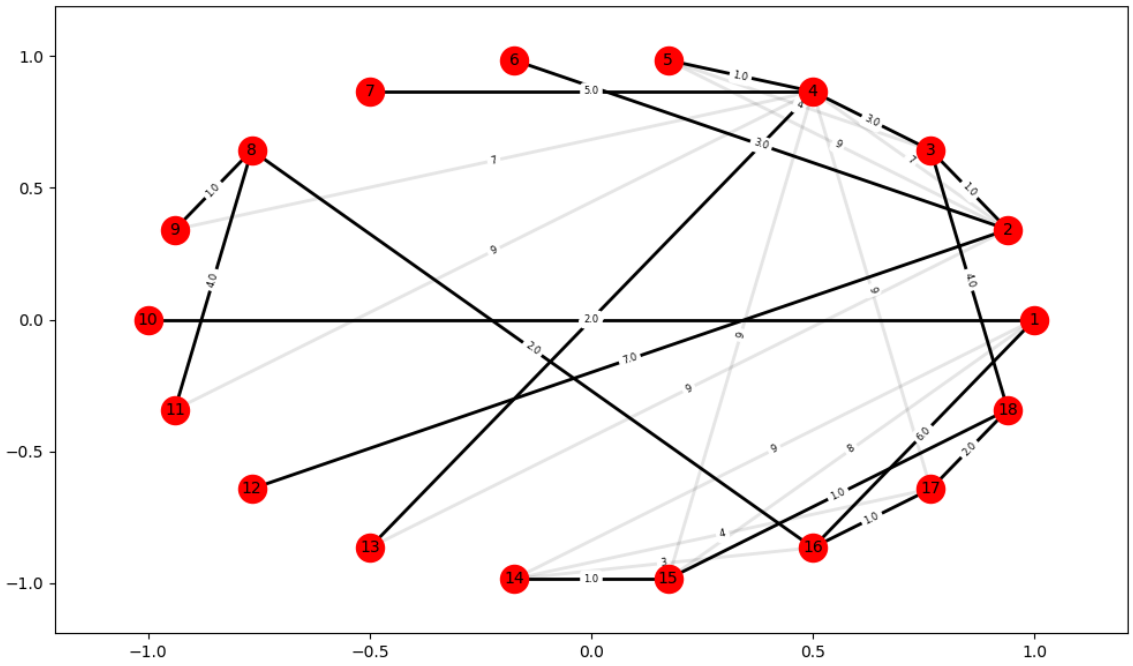
0.0	0.0	7.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0
7.0	1.0	0.0	8.0	3.0
0.0	0.0	8.0	0.0	0.0
0.0	0.0	3.0	0.0	0.0

4.3 Prueba3

Grafo: 18 vertices, 29 aristas
Matriz entrante:

0	0	0	0	0	0	0	0	0	2	0	0	0	9	8	6	0	0
0	0	1	7	9	3	0	0	0	0	0	7	9	0	0	0	0	0
0	1	0	3	4	0	0	0	0	0	0	0	0	0	0	0	0	4
0	7	3	0	1	0	5	0	7	0	9	0	3	0	9	0	9	0
0	9	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	4	0	0	0	0	2	0	0
0	0	0	7	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	9	0	0	0	4	0	0	0	0	0	0	0	0	0	0
0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	9	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	4	0
8	0	0	9	0	0	0	0	0	0	0	0	0	1	0	0	0	1
6	0	0	0	0	0	0	2	0	0	0	0	0	3	0	0	1	0
0	0	0	9	0	0	0	0	0	0	0	0	0	4	0	1	0	2
0	0	4	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0

Matriz y grafica resultante:



0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0
0.0	0.0	3.0	0.0	1.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	4.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
6.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	2.0
0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	2.0	0.0

4.4 Siguientes pruebas

El siguiente link lleva a un txt en donde se encuentran los resultados de las 10 Pruebas restantes, cada prueba posee un link a la imagen que le conrresponde.
Pruebas restantes.

Bibliografía

[1] NetworkX Documentation

[2] NetworkX Reference

[3] Tkinter 8.5 reference: a GUI for Python

[4] Matplotlib user's Guide

[5] Matriz Numpy