Practica_1_Sara_Dorado_Alfaro_Alvaro_Huertas_Garcia

March 2, 2020

1 PRÁCTICA 1

1.1 Autores:

- Sara Dorado Alfaro
- Álvaro Huertas García

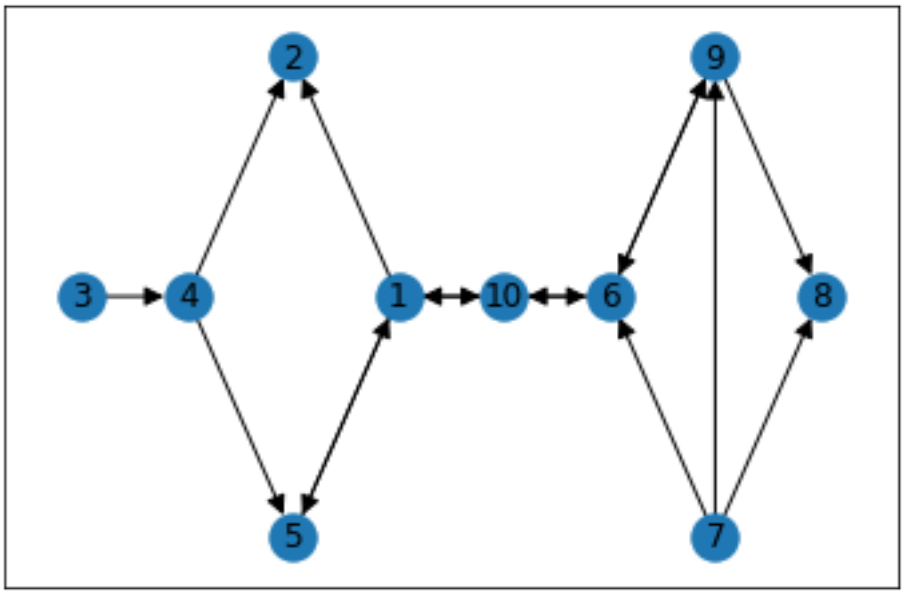
1.2 Fecha: 06/02/2020

1.3 Apartado 1

[1]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
```

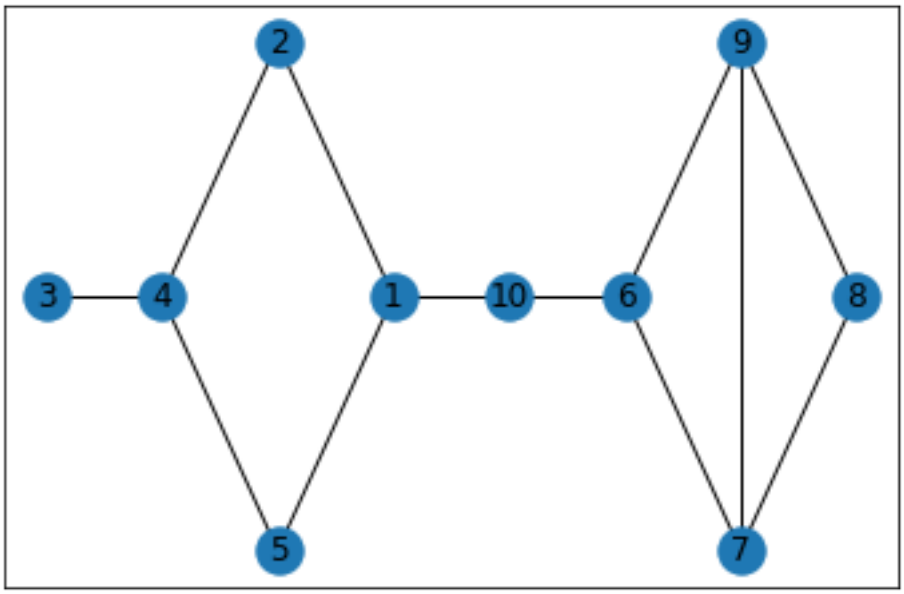
[41]:
```
# Mostramos el grafo dirigido de la práctica
G = nx.DiGraph()
G.add_nodes_from(['1', '2', '5', '10', '3', '4', '6', '9', '7', '8'])
G.add_edges_from([('1', '2'), ('1', '5'), ('5', '1'), ('1', '10'), ('4', '2'),␣
 ↪('4', '5'),
                  ('10', '6'), ('10', '1'), ('6', '10'), ('3', '4'),
                  ('6', '9'), ('7', '6'), ('7', '9'), ('9', '8'), ('9', '6'),␣
 ↪('7', '8')])

posiciones = {'1': [7, 0], '2':[5, 1],
              '5': [5, -1], '10': [9, 0],
              '3': [1, 0], '4': [3, 0],
              '6': [11, 0], '9': [13,  1],
              '7': [13, -1], '8': [15, 0]}

print("Grafo dirigido")
nx.draw_networkx(G, pos = posiciones, arrowsize = 15, edge_color = 'black')
```

```
Grafo dirigido
```



[3]:
```
# Mostramos el grafo no dirigido

G = nx.Graph()
G.add_nodes_from(['1', '2', '5', '10', '3', '4', '6', '9', '7', '8'])
G.add_edges_from([('1', '2'), ('1', '5'), ('1', '10'), ('2', '4'), ('5', '4'),␣
 ↪('10', '6'), ('3', '4'),
                  ('6', '9'), ('6', '7'), ('9', '7'), ('9', '8'), ('7', '8')])

posiciones = {'1': [7, 0], '2':[5, 1],
              '5': [5, -1], '10': [9, 0],
              '3': [1, 0], '4': [3, 0],
              '6': [11, 0], '9': [13,  1],
              '7': [13, -1], '8': [15, 0]}

print("Grafo no dirigido")
nx.draw_networkx(G, pos = posiciones, with_label = True)
```

```
Grafo no dirigido
```



1.3.1 1. Representad el siguiente grafo dirigido mediante (a) una matriz de adyacencia y (b) una lista de adyacencia.

a) Matriz de adyacencia (grafo dirigido):

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

b) Lista de adyacencia:

$1 \rightarrow 2 \rightarrow 5 \rightarrow 10$

2

$3 \rightarrow 4$

$4 \rightarrow 2 \rightarrow 5$

$5 \to 1$
$6 \to 9 \to 10$
$7 \to 6 \to 8 \to 9$
8
$9 \to 8$
$10 \to 1 \to 6$

1.3.2 2. Responded a las siguientes preguntas:

Notación: v_i es el vértice i-ésimo del grafo.

a) ¿Es ponderado?
No, porque las aristas no tienen peso.

b) ¿Es conexo?
No, porque no hay camino entre cualquier par de vértices. Por ejemplo no hay camino entre v_7 y v_9.

c) ¿Es débilmente conexo?
Sí, porque al transformar el grado dirigido en uno no dirigido observamos que es conexo, es decir, existe un camino entre cualquier par de vértices en el grafo no dirigido.

d) ¿Cuál es su tamaño y su orden?
Tamaño $|E| = 16$, orden $|V| = 10$.

e) ¿Tiene algún punto de articulación? En caso positivo, indica cual
Para el grafo dirigido todos los nodos lo son, porque el grafo no es conexo. Para el grafo no dirigido, que sí es conexo, los puntos de articulación son: v_1, v_4, v_6, v_{10}

f) ¿Tiene lazos?
No, ningún vértice se conecta consigo mismo.

g) ¿El grafo tiene algún ciclo?
Teniendo en cuenta que un ciclo es todo paseo cerrado (el nodo de inicio y fin es el mismo) en el que no se repiten ramas ni vértices, podemos decir que sí existen ciclos en el grafo dirigido:
Sí, hay varios: $v_5 \to v_1 \to v_5 \ldots$
Ciclo 1: $C = \{v_5, v_1, v_5\} \quad k = 2$
Ciclo 2: $C = \{v_1, v_{10}, v_1\} \quad k = 2$
Ciclo 3: $C = \{v_{10}, v_6, v_{10}\} \quad k = 2$

Ciclo 4: $C = \{v_6, v_9, v_6\} \quad k = 2$
donde k indica la longitud del camino.

h) ¿Existe algún camino entre los nodos 4 y 7? En caso positivo, indica cual es y su longitud
No, porque v_7 es una fuente, así que no podemos alcanzarlo desde otro nodo.

i) ¿Existe algún camino entre los nodos 3 y 9? En caso positivo, indica cual es y su longitud
Sí:
$C = \{v_3, v_4, v_5, v_1, v_{10}, v_6, v_9\} \quad k = 6$

Considera ahora el grafo como un grafo no dirigido:
a) ¿El grafo tiene algún ciclo? En caso positivo, indica cual
Teniendo en cuenta que un ciclo es todo paseo cerrado (el nodo de inicio y fin es el mismo) en el que no se repiten ramas ni vértices, podemos decir que sí existen ciclos en el grafo no dirigido:
Ciclo 1: $C = \{v_4, v_2, v_1, v_5, v_4\}$
Ciclo 2: $C = \{v_6, v_9, v_8, v_7, v_6\}$
Ciclo 3: $C = \{v_6, v_9, v_7, v_6\}$
Ciclo 4: $C = \{v_8, v_7, v_9, v_8\}$

b) ¿Cuál es el mayor valor de k para el cual existe un k-core?
Un k-core de un grafo G es un subgrafo G' tal que el grado de cada nodo G' es al menos k. Por lo tanto,en el grafo hay dos 2-core:

- $G'_1 = (V'_1, E'_1)$ donde $V'_1 = \{v_6, v_7, v_9\}$ y $E'_1 = \{(v_6, v_7), (v_7, v_9), (v_9, v_6\}$
- $G'_2 = (V'_2, E'_2)$ donde $V'_2 = \{v_7, v_8, v_9\}$ y $E'_2 = \{(v_7, v_8), (v_7, v_9), (v_8, v_9\}$

c) ¿Cuál es el índice de clusterización de v_{10}?
El cálculo del índice de clusterización se realiza mediante la siguiente ecuación:

$$Cv = \frac{Nv}{\frac{kv(kv-1)}{2}}$$

donde Nv es el número de ramas que hay entre los vecinos del nodo v; kv es el número de vecinos del nodo v

v_{10} tiene 2 vecinos (v_1, v_6), y éstos no se conectan. Por lo tanto, el índice de clusterización de v_10 es 0. Que el índice de clusterización de v_{10} sea 0 indica que es importante y un punto débil del grafo, puesto que sin él, sus nodos vecinos quedan desconectados.

d) Calcula el camino característico del nodo 10
El cálculo del camino característico se realiza mediante la siguiente ecuación:

$$L_v = \frac{\sum_{k=1}^{|V|} d(v, v_k)}{|V| - 1}$$

donde $d(v, v_k)$ es la distancia de v a un nodo v_k del grafo; $|V|$ es el orden del grafo.
Camino característico de v_{10}.

$$\frac{1+2+4+3+2+1+2+3+2}{10-1} = \frac{20}{9}$$

e) ¿Existe algún cliqué de orden mayor de 2? En caso positivo, indica los nodos que lo componen

Un cliqué es un conjunto de vértices en el que todo par de vértices distintos son adyacentes, es decir, existe una arista que los conecta. En otras palabras, un clique es un subgrafo en el que cada vértice está conectado a todos los demás vértices del subgrafo (a.k.a. completo).

En el grafo no dirigido existen dos cliqués (precisamente los dos 2-cores). Los subgrafos:

- $G_1' = (V_1', E_1')$ donde $V_1' = \{v_6, v_7, v_9\}$ y $E_1' = \{(v_6, v_7), (v_7, v_9), (v_9, v_6\}$
- $G_2' = (V_2', E_2')$ donde $V_2' = \{v_7, v_8, v_9\}$ y $E_2' = \{(v_7, v_8), (v_7, v_9), (v_8, v_9\}$

No hay cliqués de orden mayor que 3.

1.4 Apartado 2: Análisis de una red de interacción de proteínas medianteNetworkX.

1. Descargad de Moodle el grafo *CaernoElegans-LC_uw.txt*, el grafo contiene una red de interacción de proteínas correspondiente al gusano *Caernobidis Elegans*

2. El fichero que contiene la red está en formato lista de ramas, por tanto, cargad el grafo en una variable G_CE mediante la función read_edgelist("CL-LC_uw.txt").

[4]:
```
G_CE = nx.read_edgelist("CaernoElegans-LC_uw.txt")
G_CE.name = 'CaernoElegans' # añadimos atributo nombre
```

[5]:
```
### Exportar a gexf para visualizar
nx.write_gexf(G_CE, "CaernoElegans.gexf")
```

Exportamos la red a formato `.gexf` para su posterior visualización con la herramienta `gephi`. Algoritmo de visualización: Fruchterman Reingold. Claramente podemos observar algunos nodos clave, con un grado muy elevado. Este análisis se ejecutará con más detalle en las próximas secciones de la práctica.

3. Obtened e imprimid por la salida el orden y el tamaño del grafo

[6]:
```
## Orden y tamaño
tam =  nx.number_of_edges(G_CE) # numero de aristas
orden = nx.number_of_nodes(G_CE) # numero de vertices
```

```
print ("Tamaño:" , tam)
print ("Orden:", orden )
```

```
Tamaño: 1648
Orden: 1387
```

También podría haberse obtenido a partir de la información que aporta la función "info()" ded NetworkX

[7]:
```
print(nx.info(G_CE))
```

```
Name: CaernoElegans
Type: Graph
Number of nodes: 1387
Number of edges: 1648
Average degree:   2.3764
```

Averiguad si el grafo es dirigido o no

Entre las funciones que incorpora NetworkX se encuentra "is_directed()" que devuelve True en caso del que sea dirigido y False en caso contrario

[8]:
```
print(nx.is_directed(G_CE))
```

```
False
```

Por tanto, el grafo cargado perteneciente a *Caernobidis elegans* es no dirigido.

¿Es un grafo denso o disperso?

La dispersión de un grafo se calcula con la siguiente fórmula:

$$D = \frac{2|E|}{|V|(|V|-1)}$$

Donde:

- $|E|$ es el número de aristas
- $|V|$ es el número de vértices

[9]:
```
densidad = (2*tam)/(orden*(orden-1))
print(round(densidad, 5))
```

```
0.00171
```

También se puede emplear la función "density()" de NetworkX

[10]:
```
densidad_2 = nx.density(G_CE)
print(round(densidad_2, 5))
```

```
0.00171
```

Al ser muy cercano a 0 podemos asegurar que se trata de un grafo disperso. El hecho de se trate de un grafo disperso es bueno y lo hace interesante de estudiar. Por un lado, los algoritmos empleados en cálculos de parámetros de grafos proceden mucho más rápido en grafos dispersos que en grafos densos. Igualmente, las redes biológicas son en la mayor parte de las ocasiones grafos dispersos, lo que las hace muy interesante desde el punto de vista de su estudio.

4. Cread un grafo aleatorio G_AL que tenga el mismo orden y tamaño que el grafo que acabáis de cargar mediante la función gnm_random_graph(n,m)

Esta función nos permite generar un grafo aleatorio con **n** nodos y **m** ramas por nodo.

[11]:
```
# Reutulizamos orden y tamaño calculado previamente
G_AL = nx.gnm_random_graph(orden, tam, seed = 1)
G_AL.name = "Aleatorio" # añadimos el atributo nombre

# Comprobamos que las caracteristicas sean iguales
print(nx.info(G_AL))
```

```
Name: Aleatorio
Type: Graph
Number of nodes: 1387
Number of edges: 1648
Average degree:   2.3764
```

Igualemente, podemos conocer la probabilidad con la que se ha generado el grafo, igualando las dos siguientes fórmulas:

$$< k > = \frac{2|E|}{N} \quad (1)$$

$$|E| \simeq p \cdot \frac{N \cdot (N-1)}{2} \quad (2)$$

$$\frac{k \cdot N}{2} \simeq p \cdot \frac{N \cdot (N-1)}{2} \rightarrow p \simeq \frac{< k >}{N-1} \simeq 0.0017$$

Donde:

- $|E|$ es el número de aristas
- $< k >$ es el grado medio del grafo
- N es el número d vértices del grafo
- p es la probabilidad de que una rama esté presente en el grafo aleatorio

REPETIMOS LOS CÁLCULOS PARA 20 GRAFOS ALEATORIOS Con el fin de realizar comparaciones justas. Esto se hace para tener valores robustos de parámetros como el camino característico o el coeficiente de clusterización. Después se comprobará si los valores obtenidos para el grafo CaernoElegans y grafos aleatorios con el mismo orden y tamaño son distintos de manera significativa, teniendo en cuenta la media y la desviación estándar.

[12]:
```
# REPETIMOS LOS CÁLCULOS PARA 20 GRAFOS ALEATORIOS

# Creamos las listas contenedoras de los resultados de diferentes iteraciones
l_centralidad = []
l_cercania = []
l_nodo_max_centralidad = []
l_nodo_max_cercania = []
l_betweenness = []
l_nodo_max_bt = []
l_clustering = []
l_nodo_max_clust = []
l_kcores = []
l_min_path = []
l_num_comp = []
l_diameter = []

# Generamos un bucle para poder hacer métricas estadísticas de los parámetros␣
 ↪del grafo aleatorio
for i in range(0, 20):
    g = nx.gnm_random_graph(orden, tam, seed=i)

    # Centralidad
    l_centralidad.append(sum(nx.degree_centrality(g).values()))
    l_nodo_max_centralidad.append(max(nx.degree_centrality(g).items(), key =␣
 ↪lambda pareja: pareja[1])[1])

    # Cercanía
    l_cercania.append(sum(nx.closeness_centrality(g).values()))
    l_nodo_max_cercania.append(max(nx.closeness_centrality(g).items(), key =␣
 ↪lambda pareja: pareja[1])[1])

    # Betweenness
    l_betweenness.append(sum(nx.betweenness_centrality(g).values()) / orden)
    l_nodo_max_bt.append(max(nx.betweenness_centrality(g).items(), key = lambda␣
 ↪pareja: pareja[1]))

    # Indice de clusterizacion
    l_clustering.append(nx.average_clustering(g))
    l_nodo_max_clust.append(max(nx.clustering(g).items(), key = lambda pareja:␣
 ↪pareja[1])[1])
```

```
    # max k-core
    l_kcores.append(max(nx.core_number(g).values()))

    # Camino caracteristico
    l_min_path.append(nx.average_shortest_path_length(max(nx.
 ↪connected_component_subgraphs(g), key = len)))

    # Numero de componente y diametro de la componente gigante
    l_num_comp.append(nx.number_connected_components(g))
    l_diameter.append(nx.diameter(max(nx.connected_component_subgraphs(g), key =␣
 ↪len)))



# Guardamos los resultados de la media y des. std en un diccionario para acceder␣
 ↪posteriormente
dic_params = {}

# Centralidad
dic_params["Degree Centrality"] = np.mean(np.array(l_centralidad)), np.std(np.
 ↪array(l_centralidad))
dic_params["Nodo max Degree Centrality"] = np.mean(np.
 ↪array(l_nodo_max_centralidad)), np.std(np.array(l_nodo_max_centralidad))

# Cercanía
dic_params["Cercania"] = np.mean(np.array(l_cercania)), np.std(np.
 ↪array(l_cercania))
dic_params["Nodo max cercania"] = np.mean(np.array(l_nodo_max_cercania)), np.
 ↪std(np.array(l_nodo_max_cercania))

# Betweenness
dic_params["Betweenness"] = np.mean(np.array(l_betweenness)), np.std(np.
 ↪array(l_betweenness))
dic_params["Nodo max betweenness"] = np.mean(np.array(l_nodo_max_bt)), np.std(np.
 ↪array(l_nodo_max_bt))

# Indice de clusterizacion
dic_params["Clustering"] = np.mean(np.array(l_clustering)), np.std(np.
 ↪array(l_clustering))
dic_params["Nodo max clustering"] = np.mean(np.array(l_nodo_max_clust)), np.
 ↪std(np.array(l_nodo_max_clust))

# max k-core
dic_params["Max k-core"] = np.mean(np.array(l_kcores)), np.std(np.
 ↪array(l_kcores))
```

```
# Camino caracteristico
dic_params["Shortest path"] = np.mean(np.array(l_min_path)), np.std(np.
 ↪array(l_min_path))

# Numero de componente y diametro de la componente gigante
dic_params["Componentes"] = np.mean(np.array(l_num_comp)), np.std(np.
 ↪array(l_num_comp))
dic_params["Diámetro"] = np.mean(np.array(l_diameter)), np.std(np.
 ↪array(l_diameter))

# Visualizamos los resultados de los parámetros.
print(dic_params)
```

```
{'Degree Centrality': (2.378066378066355, 1.9100999153570945e-15), 'Nodo max
Degree Centrality': (0.006493506493506494, 0.0006453298636363029), 'Cercania':
(135.34516157436457, 2.3001931602143246), 'Nodo max cercania':
(0.1455585998306277, 0.0038772866573120023), 'Betweenness':
(0.003852702791742197, 0.0001306928874608582), 'Nodo max betweenness':
(328.47126017957453, 425.60895451056115), 'Clustering': (0.0009396727601486074,
0.001109340196485008), 'Nodo max clustering': (0.4133333333333334,
0.4019950248448356), 'Max k-core': (2.0, 0.0), 'Shortest path':
(7.9675033029459525, 0.10250774207511149), 'Componentes': (146.55,
7.651633812461231), 'Diámetro': (19.1, 1.337908816025965)}
```

5. Indica si ambos grafos son conexos

Según la definición de conexidad, un grafo es conexo si para cada par de nodos del grafo existe al menos un camino que los une (paseo de un un nodo u a un nodo v en el cual todos los vértices $\{v_0, v_1, ..., v_k\}$ son distintos).

Esta característica del grafo puede conocerse a partir de la función "is_connected()" de NetworkX

[13]:
```
print("Aleatorio (G_AL) -->", nx.is_connected(G_AL),
      "\nNúmero de componentes (G_AL):", nx.number_connected_components(G_AL),
      "\n\nGrafo inicial (G_CE) --> ", nx.is_connected(G_CE),
      "\nNúmero de componentes (G_CE):", nx.number_connected_components(G_CE))
```

```
Aleatorio (G_AL) --> False
Número de componentes (G_AL): 142

Grafo inicial (G_CE) -->  False
Número de componentes (G_CE): 89
```

Ninguno de los grafos es conexo.

6. ¿Cuál es el nodo con mayor grado en cada grafo?

El grado de los nodos de un grafo (número de vecinos adyacentes que posee cada nodo) se puede conocer con el atributo "G.degree()" del objeto creado por NetworkX.

[14]:
```
## Generamos un diccionario con el nodo como clave y el grado como valor
degree_dic = dict(G_CE.degree())

## Generamos una tupla con la pareja clave-valor, y extraemos la pareja con␣
 ↪mayor valor (grado)
print(max(degree_dic.items(), key = lambda k: k[1]))
```

```
('T08G11.5', 131)
```

Encontramos la proteína T08G11.5 con un grado muy alto (131). Esto puede indicar que es una proteína involucrada en un gran número de procesos.

7. ¿Cuál es la máxima distancia entre dos nodos del grafo (diámetro del grafo)?

El diámetro del grafo puede obtenerse con la función "diameter()" de NetworkX.

[15]:
```
try:
    print(nx.diameter(G_CE)) # If the diameter does not exit give an error show␣
 ↪it

except:
    print("El diámetro del grafo no se puede conocer dado que se trata de un␣
 ↪grafo no conexo")
    diametro_G_CE = nx.diameter(max(nx.connected_component_subgraphs(G_CE), key␣
 ↪= len))
    print("En su lugar se muestra el diámetro de la mayor componente del grafo␣
 ↪CaernoElegans:", diametro_G_CE)
```

```
El diámetro del grafo no se puede conocer dado que se trata de un grafo no
conexo
En su lugar se muestra el diámetro de la mayor componente del grafo
CaernoElegans: 22
```

Dado que en el grafo aleatorio el grado medio es superior a 1 ($< k >> 1$) aparece un cluster gigante, siendo el diámetro del clúster gigante el diámetro del grafo aleatorio.

[16]:
```
print("El diámetro promedio de la componente gigante del grafo aleatorio es:",␣
 ↪dic_params["Diámetro"][0], "con una desviación estándar de",
     dic_params["Diámetro"][1])
```

```
El diámetro promedio de la componente gigante del grafo aleatorio es: 19.1 con
una desviación estándar de 1.337908816025965
```

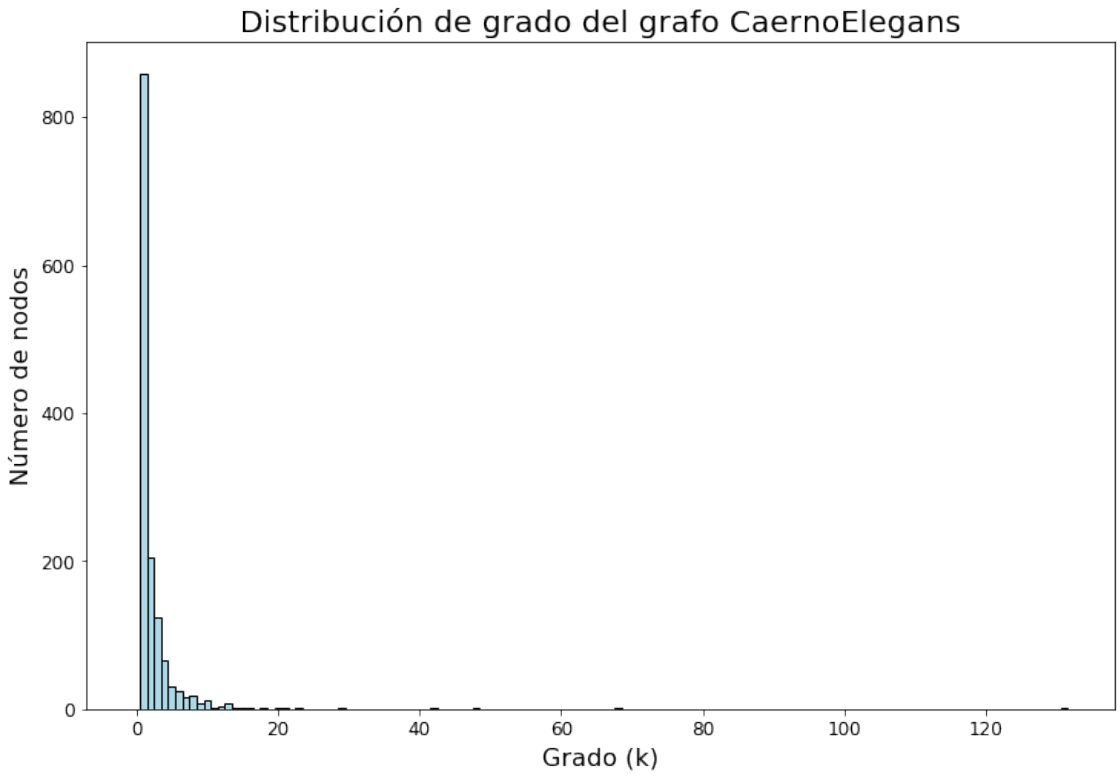
El diametro es interesante conocerlo porque es una cota superior al camino característico, puesto que el camino característico nunca puede ser superior al diámetro al igual que una media no puede ser superior al mayor valor de la muestra. Por lo tanto, si el diámetro es pequeño se comprobará que el camino característico es pequeño.

1.5 Apartado 3: Distribución del grado de los nodos

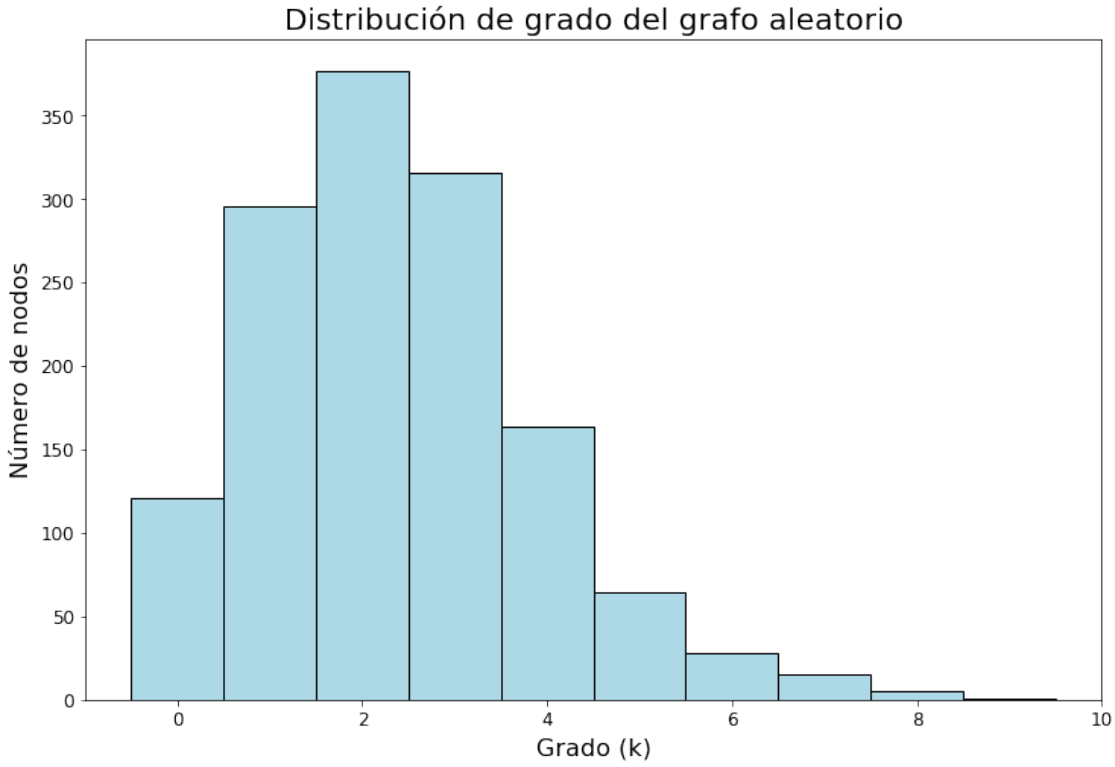
1. Visualizad la distribución del grado de los nodos de ambos grafos

Primero mostramos la distribución para el grafo CaernoElegans y después para un grafo aleatorio de mismo orden y magnitud.

[17]:
```
## Distribucion del grado de los nodos del grafo inicial
L=nx.degree_histogram(G_CE)

# tamaño figura
plt.figure(figsize = (12, 8))
plt.bar(range(len(L)),L, color='lightblue', width = 1, edgecolor = "black",␣
 ↪align = 'center')

# titulo
plt.title("Distribución de grado del grafo CaernoElegans", fontsize = 20)

# ejes X y Y
plt.ylabel("Número de nodos", fontsize = 16)
plt.xlabel("Grado (k)", fontsize = 16)
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)

plt.show()
```



[18]:
```
## Distribucion del grado de los nodos del grafo aleatorio
L=nx.degree_histogram(G_AL)

# tamaño figura
plt.figure(figsize = (12, 8))
plt.bar(range(len(L)),L, color='lightblue', width = 1, edgecolor = "black",␣
 ↪align = 'center')

# titulo
plt.title("Distribución de grado del grafo aleatorio", fontsize = 20)

# ejes X y Y
plt.ylabel("Número de nodos", fontsize = 16)
plt.xlabel("Grado (k)", fontsize = 16)
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)

# Dibujamos el grado medio

plt.show()
```



[19]:
```
print(nx.info(G_CE))
print()
print(nx.info(G_AL))
```

```
Name: CaernoElegans
Type: Graph
Number of nodes: 1387
Number of edges: 1648
Average degree:   2.3764

Name: Aleatorio
Type: Graph
Number of nodes: 1387
Number of edges: 1648
Average degree:   2.3764
```

2. ¿Son iguales las gráficas de distribución de grados de ambos grafos? ¿Qué conclusión sacas de lo anterior?

Claramente se observa que ambos gráficos no siguen la misma distribución del grado de sus nodos, aunque ambos poseen el mismo número de nodos y de ramas y grado medio.

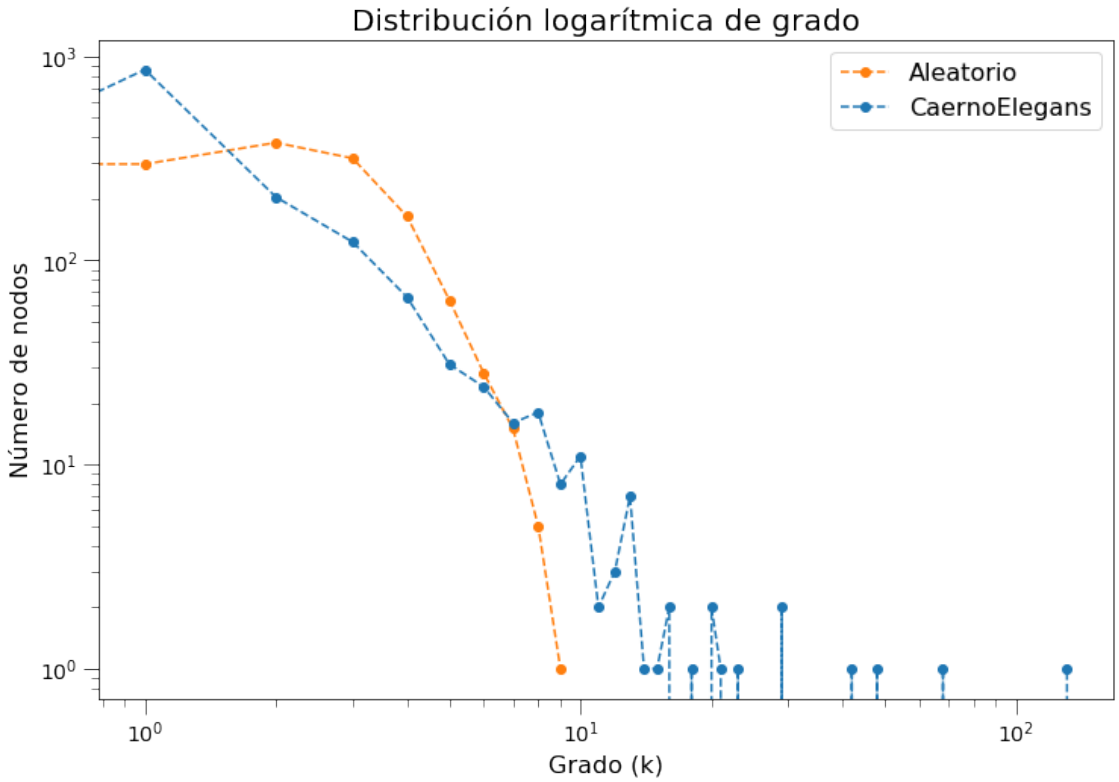
En el caso del grafo CaernoElegans vemos que hay algunos nodos con un grado alto (por ejemplo anteriormente se comprobó que el nodo con mayor grado presentaba un grado de 131) y un gran número de nodos con grado bajo, que en el caso del grafo aleatorio no se observa. Este comportamiento demuestra que nuestro grafo inicial no se comporta como un grafo aleatorio y, por tanto, podemos asegurar que no se trata de un grafo aleatorio.

3. Dibuja ahora la distribución del grado de los nodos de la red de interacción de proteínas usando escala logarítmica en ambos ejes añade para ellos dos líneas de código para cambiar el tipo de escala en cada eje:

plt.xscale("log", nonposx = 'clip')
plt.yscale("log", nonposy = 'clip')

[20]:
```
AL = nx.degree_histogram(G_AL)
CE = nx.degree_histogram(G_CE)

# Tamaño de la figura
plt.figure(figsize = (12, 8))

# Distribución de grado de grafo aleatorio
plt.plot(range(len(AL)), AL, linestyle='--', marker='o', color='tab:orange',␣
 ↪label = "Aleatorio")
# Distribución de grado de grafo de C. elegans
plt.plot(range(len(CE)), CE, linestyle='--', marker='o', color='tab:blue', label␣
 ↪= "CaernoElegans")
```

```
# Titulo
plt.title("Distribución logarítmica de grado", fontsize = 20)

# ejes X y Y
plt.ylabel("Número de nodos", fontsize = 16)
plt.xlabel("Grado (k)", fontsize = 16)
plt.xscale("log", nonposx = 'clip')
plt.yscale("log", nonposy = 'clip')
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)

# parametros de los ticks
plt.tick_params(axis="both", which = "minor", direction="out", length = 5)
plt.tick_params(axis="both", which = "major", direction="out", length = 10)

# leyenda
plt.legend(loc = "best", fontsize = 16)

plt.show()
```



4. ¿Qué tipo de gráfica obtienes? ¿Podrías calcular aproximadamente la pendiente de los datos?

La gráfica del grafo CaernoElegans tiene una región claramente lineal, fenómeno que no ocurre en la curva generada para el grafo aleatorio.

La aparición de esta región lineal es característica de una red libre de escala; donde algunos nodos están altamente conectados, mientras que el grado de conectividad de casi todos los nodos es bastante bajo. Esto además concuerda con el hecho de que la pendiente sea negativa. Tomando de forma aproximada dos puntos de la región lineal de la gráfica obtenemos:

$$pendiente = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3-1}{0-0.9} = -2.22$$

Adicionalmente, observamos que tras la región rectilínea, se forma una especie de cola correspondiente a la desaparición de nodos durante la formación de la red.

Por el contrario, en el caso de la curva asociada al grafo aleatorio, no encontramos una región lineal. Por tanto, no es posible calcular su pendiente. Esta representación curvilínea se corresponde a los grafos aleatorios, donde la frecuencia de nodos disminuye a medida que aumenta el número de grado debido a que la probabilidad de que se formen ramas es igual para todos los nodos. Esta representación curvilínea se asocia con una distribución binomial con una probabilidad de que se genere una rama (p) muy baja (en nuestro caso $p \simeq 0.0017$), es decir, una distribución de Poisson.

1.6 Apartado 4: Cálculo de parámetros de grafos

Calcula para ambos grafos:

1.6.1 a) degree_centrality(G)

Dentro de las medidas de centralidad de un grafo encontramos la centralidad del grado ("degree centrality"), que corresponde al número de ramas que posee un nodo con los demás [1]. Dado que el grafo no es un grafo dirigido, no hay que diferenciar entre grados de salida y de entrada. Por lo tanto, para conocer la centralidad se emplea la función: *degree_centrality()* de NetworkX.

Se calcula de la siguiente forma:

$$C_{DEG}^{v} = \frac{k_v}{|V|}$$

$$\bar{C}_{DEG}^{G} = \frac{\sum_{v=1}^{|V|} C_{DEG}^{v}}{|V|}$$

Donde:

- C_{DEG}^{v} es la centralidad del grado del nodo v
- $\bar{C}_{DEG}$ es el promedio de la centralidad del grado del grafo G
- k_v es el grado del nodo v
- $|V|$ es el orden del grafo

La centralidad del grado nos permite conocer como de conectado se encuentra un nodo con el resto de nodos. Por ejemplo, en el grafo correspondiente a la red de *C. elegans* se comprueba que el nodo con mayor grado calculado previamente es el que presenta mayor valor de centralidad del grado. De este modo, podemos determinar nodos importantes de la red y potenciales dianas para atacarla.

Observando el resultado, es lógico que ambos gráficos tengan el mismo valor promedio de centralidad del grado, porque ambos tienen el mismo número de nodos y de ramas por lo que, aunque la centralidad del grado de cada nodo individualmente sea diferente, el promedio será el mismo (la desviación es $\simeq 0$ en promedio para los grafos aleatorios generados). No obstante, sí observamos que el grafo de la red de proteínas de *C. elegans* no se comporta de manera aleatoria, dado que el valor máximo de centralidad de grado es mayor frente al máximo del grafo aleatorio ($0.09 > 0.006$).

[21]:
```
# Grafo inicial
dic_centralidad = nx.degree_centrality(G_CE) # centralidad calculada para todos␣
 ↪los nodos
# print (dic_centralidad)
# Promedio
promedio_centralidad = (sum(dic_centralidad.values()) / orden)

print("Centralidad de grado en la red de proteínas de C. elegans: ",␣
 ↪promedio_centralidad)

# Nodo con mayor valor de centralidad de grado
mayor = max(dic_centralidad.items(), key = lambda pareja: pareja[1] )
print("Nodo con mayor centralidad de grado en la red de proteínas de C. elegans:␣
 ↪%s, (%s)" %(mayor[0], mayor[1]))
```

```
Centralidad de grado en la red de proteínas de C. elegans:
0.0017145395660175464
Nodo con mayor centralidad de grado en la red de proteínas de C. elegans:
T08G11.5, (0.09451659451659451)
```

[22]:
```
print("Promedio de la centralidad media de grado en el grafo aleatorio: %s con␣
 ↪%s desviación estándar"
      %(dic_params["Degree Centrality"][0], dic_params["Degree Centrality"][1]))

print()
print("Centralidad media del nodo con mayor centralidad de grado en el grafo␣
 ↪aleatorio: %s, con desviación estándar %s"
      %(dic_params["Nodo max Degree Centrality"][0], dic_params["Nodo max Degree␣
 ↪Centrality"][1]))
```

```
Promedio de la centralidad media de grado en el grafo aleatorio:
2.378066378066355 con 1.9100999153570945e-15 desviación estándar

Centralidad media del nodo con mayor centralidad de grado en el grafo aleatorio:
0.006493506493506494, con desviación estándar 0.0006453298636363029
```

1.6.2 b) closeness_centrality(G) (cercanía)

La cercanía (inversamente proporcional a la lejanía) es otra medida de centralidad basada en el cálculo de la suma o bien el promedio de las distancias más cortas desde un nodo hacia todos los demás.

b.1) Cálculo de la cercanía en el grafo de CaernoElegans y el grafo aleatorio

[23]:
```
# Grafo inicial
dic_cercania = nx.closeness_centrality(G_CE)
promedio_cercania = (sum(dic_cercania.values()) / orden)
print("Cercanía en la red de proteínas de C. elegans: ", promedio_cercania)
print()
mayor = max(dic_cercania.items(), key = lambda pareja: pareja[1] )
print("Nodo con mayor centralidad de grado en la red de proteínas de C. elegans:␣
 ↪%s, (%s)" %(mayor[0], mayor[1]))
```

```
Cercanía en la red de proteínas de C. elegans:  0.07114268033538046

Nodo con mayor centralidad de grado en la red de proteínas de C. elegans:
W10C8.2, (0.13008480872167205)
```

[24]:
```
print("Promedio de la cercanía media de grado en el grafo aleatorio: %s, con %s␣
 ↪desviación estándar"
      %(dic_params["Cercania"][0], dic_params["Cercania"][1]))
print()
print("Centralidad media del nodo con mayor cercanía en el grafo aleatorio: %s,␣
 ↪con desviación estándar %s"
      %(dic_params["Nodo max cercania"][0], dic_params["Nodo max cercania"][1]))
```

```
Promedio de la cercanía media de grado en el grafo aleatorio:
135.34516157436457, con 2.3001931602143246 desviación estándar

Centralidad media del nodo con mayor cercanía en el grafo aleatorio:
0.1455585998306277, con desviación estándar 0.0038772866573120023
```

La cercanía se expresa como la inversa de la suma de las distancias de un grafo. Normalmente se muestra normalizada por el número de nodos diferentes al nodo de partida ($N-1$, siendo N el número total de nodos)[1]:

$$CC_i = \frac{N-1}{\sum_j d_{i,j}}$$

Donde:

- CC_i es la cercanía

- $i \neq j$
- N es el número de nodos
- $d_{i,j}$ es la distancia mínima entre el nodo i y j

Mientras mayor sea el valor de la cercanía, se puede decir que el nodo está más «cercano» al resto de nodos de la red. De este modo, la cercanía de un nodo refleja la accesibilidad de un nodo y la rapidez con la que la información se propaga en una red [1][2].

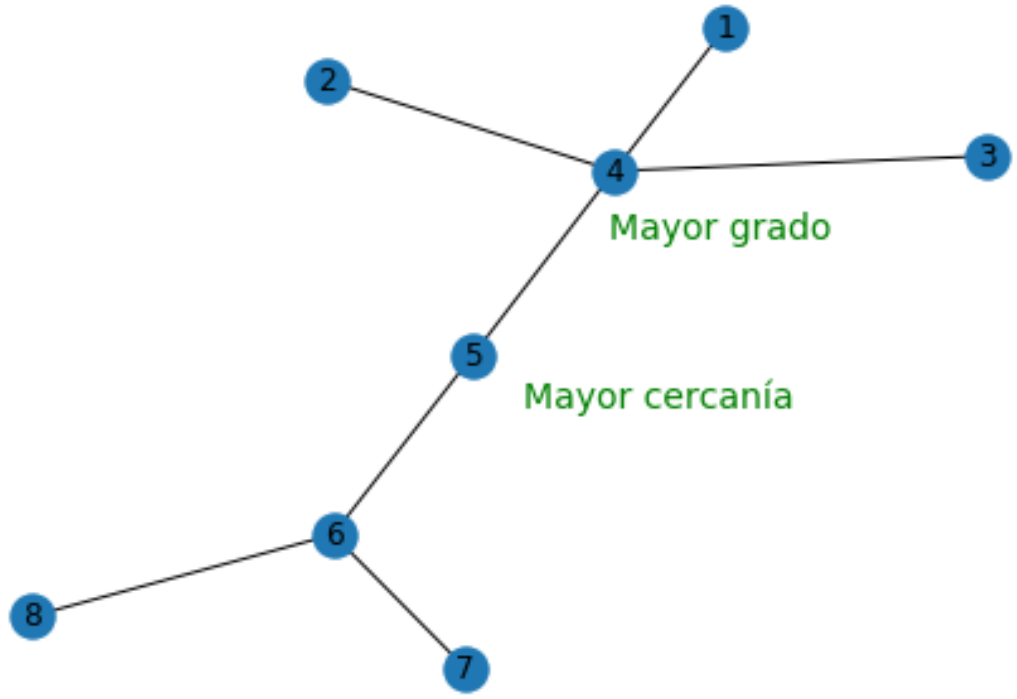
Teniendo esto último en consideración y que ambos grafos poseen el mismo número de nodos y ramas, el hecho de que el grafo aleatorio presente mayor cercanía que la red de proteínas de *C. elegans* (0.098 > 0.071) indica que las aristas del grafo CaernoElegans no se disponen del mismo modo que en los grafos aleatorios y, por tanto, que presenta un sesgo interesante de estudiar. Esto además también se observó anteriormente cuando se calculó la distribución del grado de los nodos en ambos grafos. Igualmente, el hecho de que la red de proteínas de *C. elegans* presente una cercanía menor nos indica que pueden existir nodos con un grado elevado con respecto al resto, como se ha comprobado anteriormente.

b.2) Demostración de que el nodo con mayor número de ramas no tiene porqué ser el nodo con mayor cercanía Es interesante comentar que en ambos grafos, el nodo con mayor cercanía no es el mismo que el nodo que presentaba mayor centralidad del grado. Esto es posible dado que en la cercanía se consideran las distancias más cortas desde un nodo al resto y es posible que éste no sea el que mayor número de ramas posea, como se muestra a continuación:

[25]:
```
# Ejemplo de grafo con nodo con mayor grado diferente al de mayor centralidad
H = nx.Graph()
H.add_nodes_from(['1', '2', '3', '4', '5', '6', '7', '8'])
H.add_edges_from([('1', '4'), ('2', '4'), ('3', '4'), ('4', '5'), ('5', '6'),␣
 ↪('6', '7'), ('6', '8')])

posiciones_nodos = {'1': [0.26081287, 0.8751262 ],
 '2': [-0.18917212,  0.7198457 ],
 '3': [0.55419382, 0.4984472 ],
 '4': [0.13556184, 0.45475622],
 '5': [-0.02473302, -0.0874654 ],
 '6': [-0.18162813, -0.61044471],
 '7': [-0.03313124, -1.        ],
 '8': [-0.52190402, -0.85026521]}

posiciones_labels = {'1': [0.26081287, 0.8751262 ],
 '2': [-0.18917212,  0.7198457 ],
 '3': [0.55419382, 0.4984472 ],
 '4': [0.25556184, 0.28475622],
 '5': [ 0.18473302, -0.2074654 ],
 '6': [-0.18162813, -0.58044471],
 '7': [-0.03313124, -1.        ],
 '8': [-0.52190402, -0.85026521]}
```

```
nx.draw(H, with_labels = True, pos = posiciones_nodos)
labels=nx.draw_networkx_labels(H, pos = posiciones_labels,
                               labels = {"5": "Mayor cercanía", "4": "Mayor␣
 ↪grado"},
                               font_size = 14,
                               font_color = 'green')

dic_cercania = nx.closeness_centrality(H)
promedio_cercania = (sum(dic_cercania.values()) / orden)
print("Promedio de la cercanía : ", round(promedio_cercania, 5))
```

```
Promedio de la cercanía :  0.00255
```

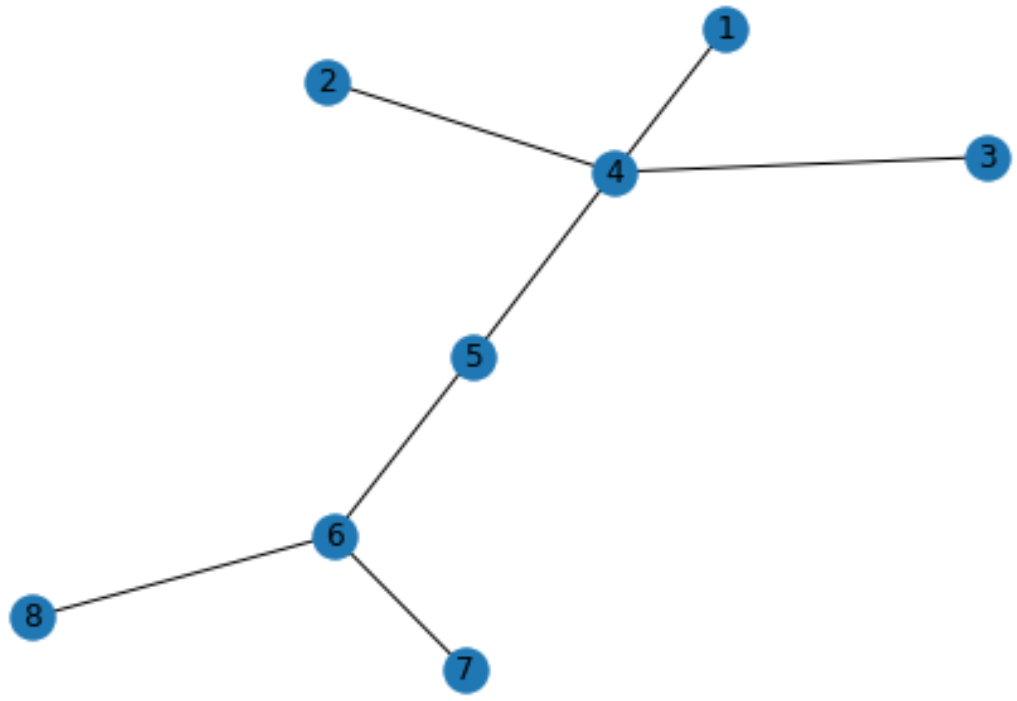


b.3) Demostración de que la cercanía depende de la distribución de las ramas en el Igualmente, se observa que la cercanía media de los dos grafos no es la misma. Esto se debe a que en el cálculo de este parámetro de centralidad, la distribución de las ramas afecta al resultado final. Por ello, a pesar de que ambos grafos presentan el mismo número de nodos y ramas, no presentan la misma distribución, lo cual afecta a los caminos mínimos que se puedan formar. Este efecto se demuestra en los dos siguientes grafos, donde la centralidad promedio cambia de 0.00255 a 0.00259 tan solo cambiando una rama (manteniéndose el número de nodos y ramas).

[26]:
```
H = nx.Graph()
H.add_nodes_from(['1', '2', '3', '4', '5', '6', '7', '8'])
```

```
H.add_edges_from([('1', '4'), ('2', '4'), ('3', '4'), ('4', '5'), ('5', '6'),␣
 ↪('6', '7'), ('6', '8')])

posiciones_nodos = {'1': [0.26081287, 0.8751262 ],
 '2': [-0.18917212,  0.7198457 ],
 '3': [0.55419382, 0.4984472 ],
 '4': [0.13556184, 0.45475622],
 '5': [-0.02473302, -0.0874654 ],
 '6': [-0.18162813, -0.61044471],
 '7': [-0.03313124, -1.        ],
 '8': [-0.52190402, -0.85026521]}

nx.draw(H, with_labels = True, pos = posiciones_nodos)

dic_cercania = nx.closeness_centrality(H)
promedio_cercania = (sum(dic_cercania.values()) / orden)
print("Promedio de la cercanía : ", round(promedio_cercania, 5))
```

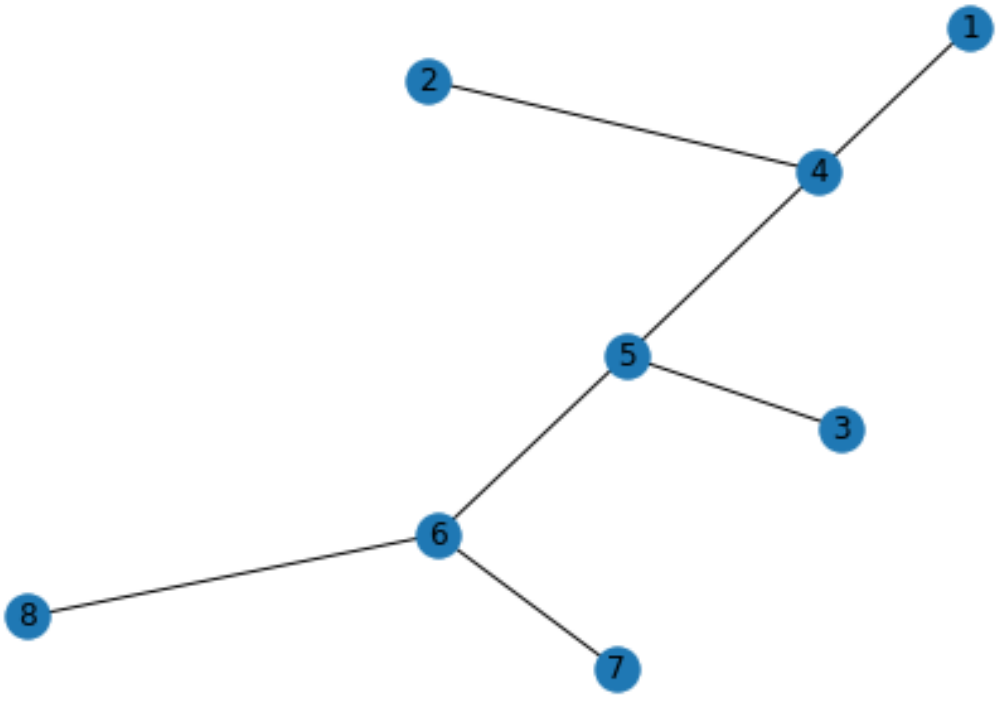
```
Promedio de la cercanía :  0.00255
```



[27]:
```
# Ejemplo de grafo con nodo con mayor grado diferente al de mayor centralidad
F = nx.Graph()
F.add_nodes_from(['1', '2', '3', '4', '5', '6', '7', '8'])
```

```
F.add_edges_from([('1', '4'), ('2', '4'), ('3', '5'), ('4', '5'), ('5', '6'),␣
 ↪('6', '7'), ('6', '8')])

posiciones_nodos = {'1': [0.26081287, 0.8751262 ],
 '2': [-0.18917212,  0.7198457 ],
 '3': [0.15419382, -0.2984472 ],
 '4': [0.13556184, 0.45475622],
 '5': [-0.02473302, -0.0874654 ],
 '6': [-0.18162813, -0.61044471],
 '7': [-0.03313124, -1.        ],
 '8': [-0.52190402, -0.85026521]}

nx.draw(F, with_labels = True, pos = posiciones_nodos)

dic_cercania = nx.closeness_centrality(F)
promedio_cercania = (sum(dic_cercania.values()) / orden)
print("Promedio de la cercanía: ", round(promedio_cercania, 5))
```

```
Promedio de la cercanía:  0.00259
```



Como conclusión podemos decir que el nodo con mayor cercanía es otra diana susceptible de ser atacada en una red, puesto que ayuda a determinar el impacto que éste causa dentro del conjunto del que forma parte [3].

1.6.3 c) beteewnness_centrality()

El beteewness de un nodo (v) se define como la fracción de caminos mínimos que hay entre el resto de nodos del grafo y que pasan por el nodo v. En otras palabras, el betweeness es una medida que cuantifica la frecuencia o el número de veces que un nodo actúa como un puente a lo largo del camino más corto entre otros dos nodos [1].

Los nodos con una alto betweeness suelen jugar un rol crítico en la estructura de la red. Esto se debe a que participan como reguladores del flujo de información en los procesos de difusión e integración de la información. De este modo, los nodos con un alto betweeness también son dianas susceptibles de ser atacadas en una red.

Aquí las ramas y nodos son los mismos pero la distribución de las ramas sí afecta al cálculo, ya que influye en los caminos mínimos que se puedan formar y a la topología de la red.

La red biológica es algo diferente a la aleatoria, siendo mayor el betweenness de la aleatoria que la de la red de proteínas. Tiene sentido que sean diferentes, dado que la red de proteínas presenta un sesgo en la distribución de sus ramas por lo que el número de caminos mínimos que pasan por un nodo individual puede ser alto, pero de media, el número de caminos mínimos que pasan por un punto será menor dado que hay clústeres, de modo que la conexión entre los clústeres tiene poco intermediarios. Mientras que en uno aleatorio los caminos mínimos que incorporen otros nodos intermediarios serán más abundantes dada la distribución homogénea de las ramas.

[28]:
```
# Grafo inicial
dic_bet = nx.betweenness_centrality(G_CE)
promedio_bet = (sum(dic_bet.values()) / orden)
print("Beteewness en la red de proteínas de C. elegans: ", promedio_bet)

mayor = max(dic_bet.items(), key = lambda pareja: pareja[1] )
print("Nodo con mayor betweeness en la red de proteínas de C. elegans: %s, (%s)"␣
 ↪%(mayor[0], mayor[1]))
```

```
Beteewness en la red de proteínas de C. elegans:  0.0025678228687606676
Nodo con mayor betweeness en la red de proteínas de C. elegans: C23G10.4,
(0.08356781880879106)
```

[29]:
```
print("Promedio del betweenness medioo en el grafo aleatorio: %s, con %s␣
 ↪desviación estándar"
      %(dic_params["Betweenness"][0], dic_params["Betweenness"][1]))
print()
print("Centralidad media del nodo con mayor betweenness en el grafo aleatorio:␣
 ↪%s, con desviación estándar %s"
      %(dic_params["Nodo max betweenness"][0], dic_params["Nodo max␣
 ↪betweenness"][1]))
```

```
Promedio del betweenness medioo en el grafo aleatorio: 0.003852702791742197, con
0.0001306928874608582 desviación estándar
```

```
Centralidad media del nodo con mayor betweenness en el grafo aleatorio:
328.47126017957453, con desviación estándar 425.60895451056115
```

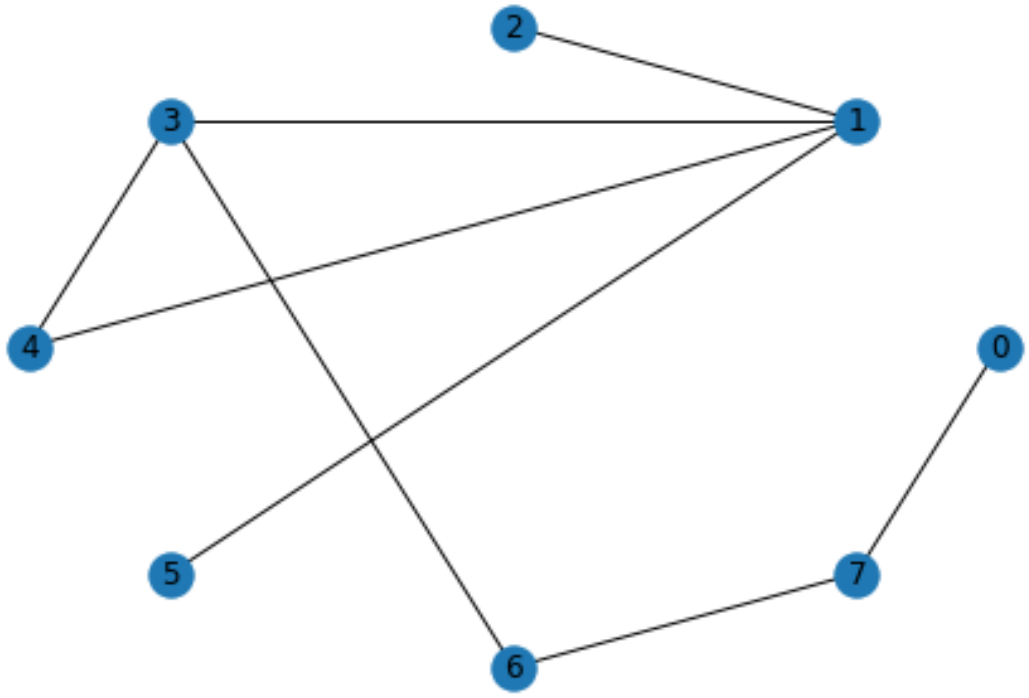
1.6.4 Ejemplos donde se compara "closeness y betweenness" de grafos aleatorios y redes de proteínas procedentes de STRING

A continuación se proponen algunos ejemplos de redes biológicas obtenidas a partir del reprositorio STRING, sobre los cuales se han calculado los valores de cercanía y betweennes, y se han generado sus respectivos grafos aleatorios de mismo orden y tamaño, con los que se han comparado. Este estudio se ha realizado para comprobar si la diferencia entre un grafo no aleatorio y un grafo aleatorio en los valores de "closeness y betweenness" presenta grandes diferencias. Nos ha parecido interesante estudiarlo, dado que el grafo CaernoElegans y el grafo aleatorio presentan distinto valores de "closeness y betweenness", pero no sabíamos qué grafo tendría que tener un valor superior o inferior con respecto al otro:

Ejemplo 1: Aleatoria vs Red ZNF480 link

[30]:
```
# Aleatoria
I = nx.gnm_random_graph(8, 8, seed = 1)

pos = {0:[1.00000000e+00, 1.83784272e-08], 1:[0.70710678, 0.70710677],
       2:[-1.73863326e-08,  9.99999992e-01], 3:[-0.70710672,  0.70710677],
       4:[-9.99999947e-01, -6.90443471e-08], 5:[-0.70710678, -0.70710667],
       6:[ 3.82499349e-08, -9.99999955e-01], 7:[ 0.70710666, -0.70710685]}

nx.draw(I, with_labels = True, pos = pos)

# Cercanía
dic_cercania = nx.closeness_centrality(I)
promedio_cercania = (sum(dic_cercania.values()) / orden)
print("Grafo aleatorio")
print("Promedio de la cercanía: ", round(promedio_cercania, 5))

# Betweenness
dic_bet = nx.betweenness_centrality(I)
promedio_bet = (sum(dic_bet.values()) / orden)
print("Promedio del betweenness:", round(promedio_bet,5))
```

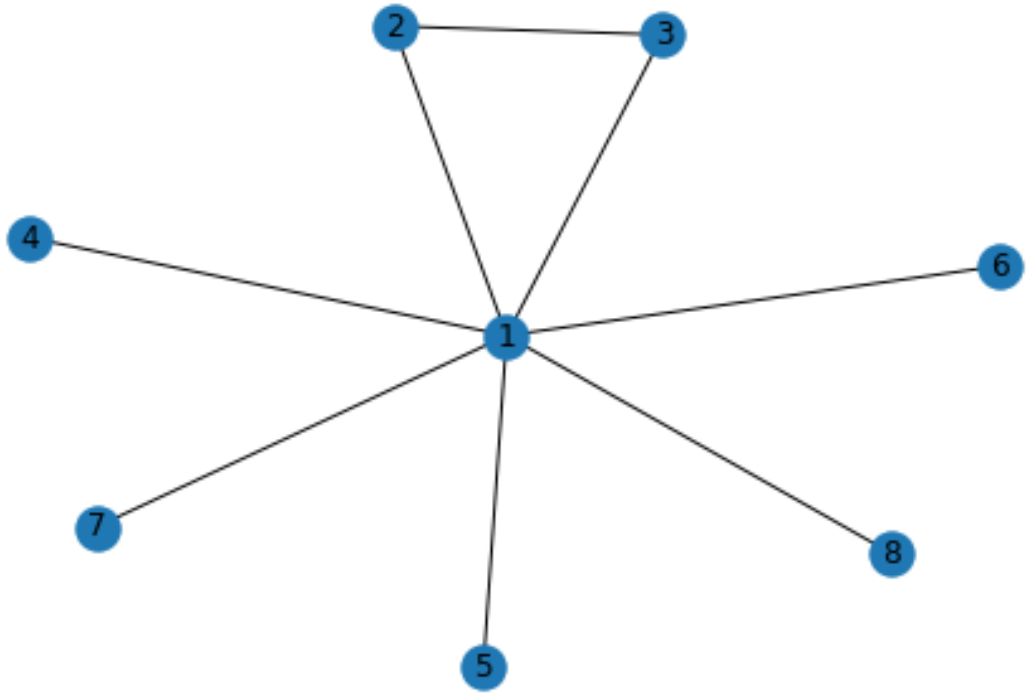
```
Grafo aleatorio
Promedio de la cercanía:  0.00253
Promedio del betweenness: 0.00134
```



[31]:
```
# Red de proteínas relacionazas con la proteína ZNF480
L = nx.Graph()
L.add_nodes_from(['1', '2', '3', '4', '5', '6', '7', '8'])
L.add_edges_from([('1', '2'), ('1', '3'), ('1', '4'), ('1', '5'), ('1', '6'),␣
 ↪('1', '7'), ('1', '8'), ('2', '3')])

pos = {'1': [-0.00162884, -0.01468591], '2': [-0.22472125,  0.90837056],
       '3': [0.30617573, 0.88310596], '4': [-0.94317228,  0.27793508],
       '5': [-0.04744948, -1.        ], '6': [0.96677375, 0.19258629],
       '7': [-0.80946116, -0.58527708], '8': [ 0.75348354, -0.66203489]}

nx.draw(L, with_labels = True, pos = pos)

print("Grafo de la proteína ZNF480 ")

# Cercanía
dic_cercania = nx.closeness_centrality(L)
promedio_cercania = (sum(dic_cercania.values()) / orden)
print("Promedio de la cercanía: ", round(promedio_cercania, 5))

# Betweenness
dic_bet = nx.betweenness_centrality(L)
promedio_bet = (sum(dic_bet.values()) / orden)
```

```
print("Promedio del betweenness:", round(promedio_bet,5))
```

```
Grafo de la proteína ZNF480
Promedio de la cercanía:  0.0035
Promedio del betweenness: 0.00069
```

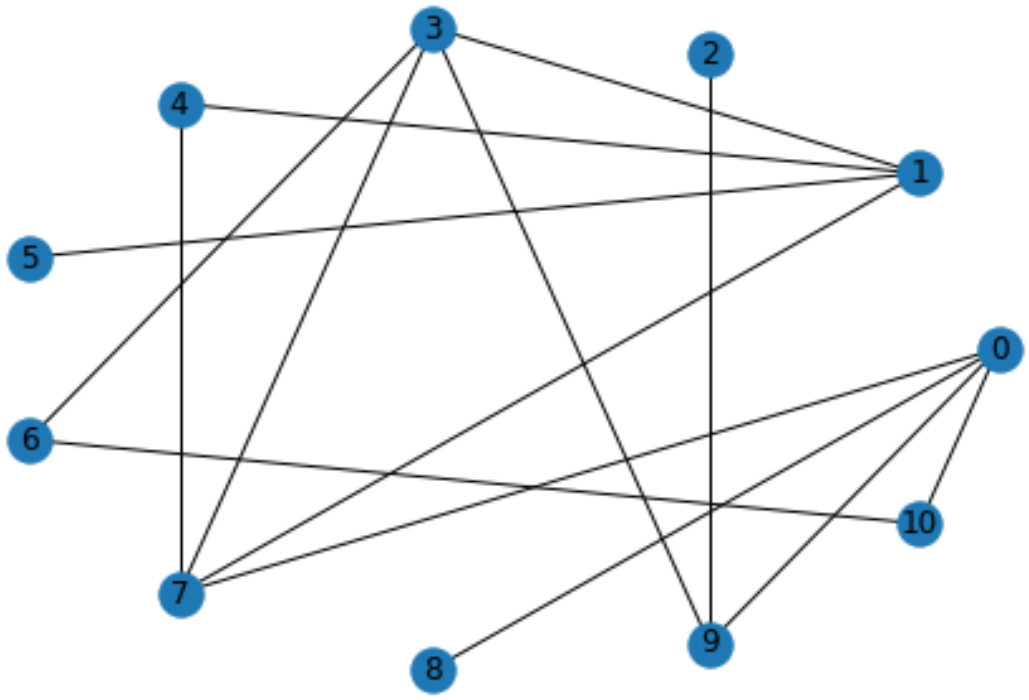


Ejemplo 2: Aleatoria vs GLIPR2 link

[32]:
```
# Aleatoria
I = nx.gnm_random_graph(11, 14, seed = 1)

pos = {0: [1., 0.], 1: [0.84125352, 0.54064077], 2: [0.41541505, 0.90963196],
       3: [-0.14231483,  0.98982143], 4: [-0.65486066,  0.75574964],
       5: [-0.95949297,  0.28173262], 6: [-0.95949297, -0.28173256],
       7: [-0.65486072, -0.75574958], 8: [-0.14231501, -0.98982143],
       9: [ 0.41541511, -0.90963196], 10: [ 0.84125346, -0.54064089]}

nx.draw(I, with_labels = True, pos = pos)
print("Grafo aleatorio")

# Cercanía
dic_cercania = nx.closeness_centrality(I)
promedio_cercania = (sum(dic_cercania.values()) / orden)
print("Promedio de la cercanía: ", round(promedio_cercania, 5))
```

```
# Betweenness
dic_bet = nx.betweenness_centrality(I)
promedio_bet = (sum(dic_bet.values()) / orden)
print("Promedio del betweenness:", round(promedio_bet,5))
```

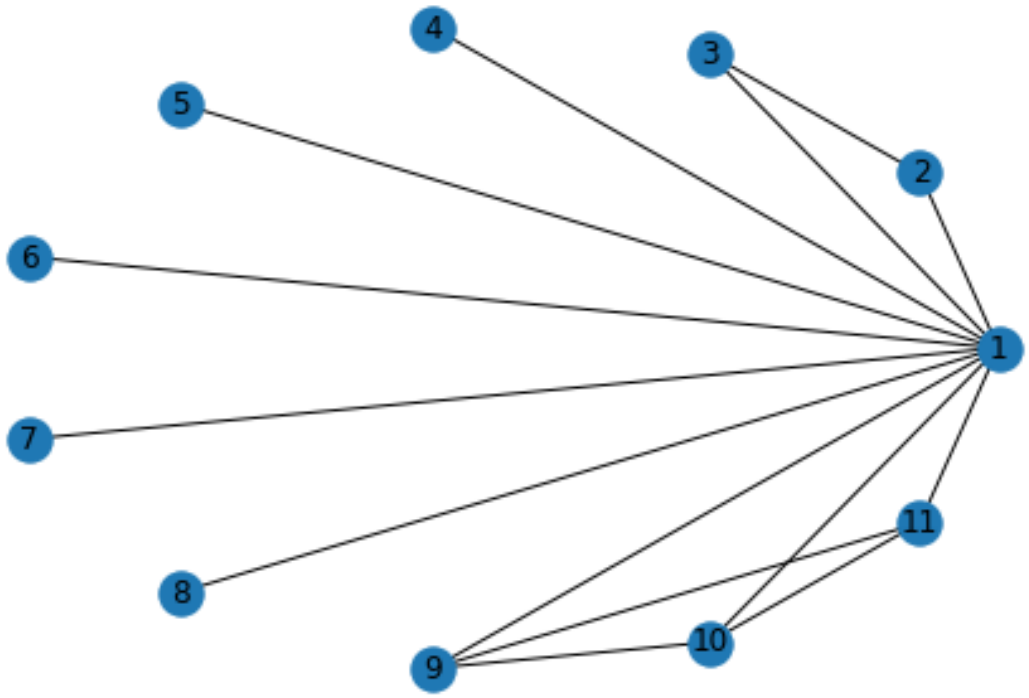
```
Grafo aleatorio
Promedio de la cercanía:  0.00377
Promedio del betweenness: 0.00104
```



[33]:
```
# Red de proteínas relacionazas con la proteína GLIPR2
L = nx.Graph()
L.add_nodes_from(['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11'])
L.add_edges_from([('1', '2'), ('1', '3'), ('1', '4'), ('1', '5'), ('1', '6'),
                  ('1', '7'), ('1', '8'), ('1', '9'), ('1', '10'), ('1', '11'),␣
 ↪('2', '3'),
                 ('9', '11'), ('10', '11'), ('9', '10')])

pos = nx.circular_layout(L)
nx.draw(L, with_labels = True, pos = pos)

print("Grafo de la proteína GLIPR2")
```

```python
# Cercanía
dic_cercania = nx.closeness_centrality(L)
promedio_cercania = (sum(dic_cercania.values()) / orden)
print("Promedio de la cercanía: ", round(promedio_cercania, 5))

# Betweenness
dic_bet = nx.betweenness_centrality(L)
promedio_bet = (sum(dic_bet.values()) / orden)
print("Promedio del betweenness:", round(promedio_bet,5))
```

```
Grafo de la proteína GLIPR2
Promedio de la cercanía:  0.00469
Promedio del betweenness: 0.00066
```



	ZNF480	Aleatorio	GLIPR2	Aleatorio	CaernoElegans	Aleatorio
Cercanía	0.0035	0.00253	0.00469	0.00377	0.071142	0.097581 ± 0.00165
Betwenness	0.00069	0.00134	0.00066	0.00104	0.002567	0.003852 ± 0.000130

1.6.5 Conclusión sobre los parámetros "clonesness y betweenness"

A raíz de los resultados obtenidos, queda claro que los grafos NO aleatorios presentan valores diferentes a sus correspondientes grafos aleatorios. El sesgo biológico que presentan los grafos de redes de proteínas provoca que los valores de "clonesness y betweenness" sean diferentes. Esta diferencia puede darse en ambas direcciones en ambos parámetros, es decir, con un valor superior o inferior de "clonesness y betweenness" con respecto al del grafo aleatorio. En conclusión, son parámetros que nos permiten conocer si existe un sesgo en nuestro grafo de estudio e identificar nodos importantes en el mismo.

1.6.6 d) average_clustering()

d.1) Cálculo del índice de clusterización El coeficiente de clusterización o agrupamiento (C) es un valor métrico local que mide el nivel de agrupamiento de los nodos. Este parámetro nos permitirá conocer cómo de agrupados se encuentran los nodos en el grafo CaernoElegans, y si existe alguna diferencia con los grafos aleatorios.

[34]:
```
# Grafo inicial
c = nx.average_clustering(G_CE)
print("Índice de clusterización de la red de proteínas de C. elegans:", c)
mayor = max(nx.clustering(G_CE).items(), key = lambda pareja: pareja[1])
print("El nodo con mayor índice de clusterización es: %s, (%s)" %(mayor[0],␣
 ↪mayor[1]))

try:
    L = nx.average_shortest_path_length(G_CE)
    print("Camino mínimo de la red de proteínas de C. elegans:", L)
except:
    print ("\nNo puede calcularse el camíno mínimo de un grafo no conexo")
```

```
Índice de clusterización de la red de proteínas de C. elegans:
0.07570841434149081
El nodo con mayor índice de clusterización es: Y38C1AA.2, (1.0)

No puede calcularse el camíno mínimo de un grafo no conexo
```

[35]:
```
# Aleatorio
print("Promedio del índice de clústering medio de grado en el grafo aleatorio:␣
 ↪%s, con %s desviación estándar"
      %(dic_params["Clustering"], dic_params["Clustering"][1]))
print()
print("Clustering media del nodo con mayor índice de clustering en el grafo␣
 ↪aleatorio: %s, con desviación estándar %s"
      %(dic_params["Nodo max clustering"][0], dic_params["Nodo max␣
 ↪clustering"][1]))
```

```
Promedio del índice de clústering medio de grado en el grafo aleatorio:
(0.0009396727601486074, 0.001109340196485008), con 0.001109340196485008
desviación estándar

Clustering media del nodo con mayor índice de clustering en el grafo aleatorio:
0.4133333333333334, con desviación estándar 0.4019950248448356
```

Una red biológica tiene un índice de clusterización alto, pero las redes aleatorias tienen un índice de clusterización bajo (dado que las ramas se distribuyen de forma aleatoria y es más difícil que aparezcan agrupaciones de nodos). De este modo, podemos comprobar de nuevo que la red de proteínas de *C. elegans* efectivamente corresponde a una red no aleatoria.

Igualmente, sería interesante conocer el camino característico de CaernoElegans para ver si se trata de una red de mundo pequeño, las cuales presentan un índice de clusterización alto (superior al de un grafo aleatorio) y un camino característico bajo (similar al de un grafo aleatorio).

Adicionalmente, se ha comprobado que en un grafo aleatorio la probabilidad de que dos vecinos de un nodo dado esten conectados es igual a la que dos nodos elegidos al azar esten conectados ($p = 0.0017$, anteriormente calculada), por lo que:

$$C_{aleatorio} \simeq p \simeq \frac{< k >}{N-1} \rightarrow 0.00094 \simeq 0.0017$$

d.2) Cálculo del camino mínimo de la componente mayor A pesar de que no se puede calcular el camino mínimo para un grafo no conexo, sí podemos conocer la componente conexa de mayor tamaño y calcular en ella su camino mínimo, observando que son muy similares entre el grafo CaernoElegans y los grafos aleatorios:

[36]:
```
# Grafo C. elegans

# Número de componentes conexas.
a = nx.number_connected_components(G_CE)
print ("Número de componentes conexas del grafo de C. elegans:", a)

# Para conocer el camino mínimo de la más grande obtengo como subgrafo la␣
 ↪componente de mayor tamaño
max_componente = max(nx.connected_component_subgraphs(G_CE), key = len)
min_path_max_componente = nx.average_shortest_path_length(max_componente)
print ("\nCamino mínimo de la componente más grande de C. elegans:",␣
 ↪min_path_max_componente)
```

```
Número de componentes conexas del grafo de C. elegans: 89

Camino mínimo de la componente más grande de C. elegans: 7.922564808498197
```

[37]:
```
print("Promedio del número de componentes conexas del grafo aleatorio: %s, con␣
 ↪%s desviación estándar"
      %(dic_params["Componentes"][0], dic_params["Componentes"][1]))
```

```
print()
print("Promedio del camino mínimo medio de grafo aleatorio: %s, con %s␣
 ↪desviación estándar"
      %(dic_params["Shortest path"][0], dic_params["Shortest path"][1]))
```

```
Promedio del número de componentes conexas del grafo aleatorio: 146.55, con
7.651633812461231 desviación estándar

Promedio del camino mínimo medio de grafo aleatorio: 7.9675033029459525, con
0.10250774207511149 desviación estándar
```

1.6.7 e) El máximo k para el cual existe un k core

Un núcleo de grado k es el máximo subgrafo en el cual todos los puntos son adyacentes a al menos otros k puntos, y fue propuesto por Seidman (1983) [4]. Este parámetro es un método muy bueno para comprobar si nuestro grado de estudio se comporta igual que los grafos aleatorios.

La medida de *k-core* apoya el resultado obtenido en el cálculo del índice de clusterización de ambos grafos. Aunque el índice de clusterización de la red de proteínas de *C. elegans* no sea muy alta (< 0.8), sí es mayor que la del grafo aleatorio (0.075708 > 0.0005827). Por ello, que el máximo k-core encontrado en la red de proteínas de *C. elegans* (k-core = 6) sea superior a la del grafo aleatorio (k-core = 2 con desv std = 0) refleja que las ramas se distribuyen en la red de proteínas de *C. elegans* de forma sesgada y no aleatoria, pudiendo concluirse de nuevo que la red de *C. elegans* no es una red aleatoria.

[38]:
```
k_core = nx.core_number(G_CE)

# Nodo con mayor valor de centralidad de grado
mayor = max(k_core.values())
print("Máximo k-core de la red de proteínas de C. elegans: %d" % mayor)
```

```
Máximo k-core de la red de proteínas de C. elegans: 6
```

[39]:
```
print("Promedio del k-core medio en el grafo aleatorio: %s, con %s desviación␣
 ↪estándar"
      %(dic_params["Max k-core"][0], dic_params["Max k-core"][1]))
```

```
Promedio del k-core medio en el grafo aleatorio: 2.0, con 0.0 desviación
estándar
```

1.7 Comparación con grafos regulares

Como hemos visto con los resultados anteriores, el grafo de CaernoElegans presenta un índice de clusterización superior un camino característico muy similar al del aleatorio. Estas características nos indican que se trata de una red de mundo pequeño. Las redes de mundo pequeño se encuentran a caballo entre redes aleatorias y redes regulares, por lo que resultaría interesante estudiar también las características de una red regular.

Para que el grafo regular sea lo más parecido a nuestro grafo de estudio de *C. elegans* se emplean los siguientes parámetros:

- $p = 0$ para que la probabilidad de formar atajos sea 0 y se genere un grafo completamente regular con el que poder comparar nuestro grafo de *C. elegans*
- $N = 1387$ para que tenga el mismo número de nodos

Se generará el grafo regular siguiendo la siguiente condición:

$$N >> k >> log(N) \rightarrow 1387 >> k >> 3.14$$

La condición $N >> k$ nos asegura que el grafo sea disperso. La condición $k >> log(N)$ nos asegura que el grafo sea conexo. Como el número de vecinos en un grafo regular tiene que ser un número entero, seleccionaremos el techo de 3.14, es decir, $k = 4$.

[40]:
```
# Emplearemos p = 0 para que el grafo sea regular y  no haya atajos
G_RE = nx.watts_strogatz_graph(1387,4,0)
print(nx.info(G_RE))
print("C:", nx.average_clustering(G_RE))
print("L:", nx.average_shortest_path_length(G_RE))
```

```
Name:
Type: Graph
Number of nodes: 1387
Number of edges: 2774
Average degree:   4.0000
C: 0.5
L: 173.75036075036076
```

Conclusiones

	CaernoElegans	Aleatorio
L	7.922564	7.967503 ± 0.102507
C	0.075708	0.000939 ± 0.0011093
Degree_Centrality	0.001714	$0.001714 \pm 1.362e\text{-}18$
Closeness_Centrality	0.071142	0.097581 ± 0.00165
Betweness_Centrality	0.002567	0.003852 ± 0.000130
Max k-core	6	2 ± 0.0
Dispersión/Densidad	0.00171	—

Con las métricas obtenidas en el apartado 2 del estudio del grafo de CaernoElegans llegamos a la conclusión que la red de proteínas del organismo *Caenorhabditis elegans* constituyen un grafo disperso, no dirigido, no conexo compuesto por 1387 nodos y 1648 ramas y con un diámetro (22) superior a los grafos aleatorios (18.5 ± 0.5).

En lo que se refiere al apartado 3, a pesar de presentar el mismo grado medio, el estudio y comparación de la distribución de grado de los nodos del grafo CaernoElegans y los grafos aleatorios nos permite llegar a la conclusión de que la red de proteínas de *Caenorhabditis elegans* no es un grafo aleatorio y sí un grafo libre de escala.

Igualmente, en la tabla superior se han recopilado los resultados de los cálculos de los distintos parámetros realizados en el apartado 4. Estos resultados apoyan el hecho de que el grafo CaernoElegans no es un grafo aleatorio y nos permiten identificar nodos importantes y susceptibles de ser atacados en la red de CaernoElegans. Por ejemplo, el nodo con mayor grado y mayor centralidad (proteína T08G11.5) es una buena diana contra la que dirigir un ataque a la red, ya que se trata de un nodo importante que se encuentra relacionada con un gran número de proteínas. Del mismo modo, las proteínas W10C8.2 y C23G10.4 presentan el mayor valor de cercanía y "betweenness", respectivamente. En consecuencia, también son nodos claves en la integración de la información en la red de proteínas y contra las que se podría dirigir un ataque. La comparación de otros cálculos, como el número de componentes conexas que conforman al grafo CaernoElegans (89) frente al grafo aleatorio (147.0 ± 5.0), y la presencia de un 6-core en la red de proteínas ausente en los grafos aleatorios (2-kore $\pm$ 0.0) corroboran que la red de proteínas presenta un sesgo que la diferencia de las redes aleatorias.

Finalmente, el estudio del índice de clusterización (C) junto con el camino característico (L) calculado para la componente conexa más grande del grafo CaernoElegans, nos permite establecer que se trata de un grafo de mundo pequeño. Esta conclusión se obtiene observando los valores recogidos en la tabla superior, donde se puede observar que el índice de clusterización del grafo CaernoElegans es superior al grafo aleatorio ($0.075708 > 0.000939 \pm 0.0011093$), pero ambas comparten el mismo camino característico ($7.922564 \simeq 7.967503 \pm 0.102507$). Este rasgo es característicos de las redes de mundo pequeño que comparten rasgos de grafos aleatorios y grafos regulares.