PARCIALITO 1

April 30, 2022

Nombre y Apellido: Manuel Longo Elia

Padron: 102425

Considerando esta red que representa las conexiones de diferentes países por los vuelos (directos) realizados entre ellos, responder las siguientes preguntas

[1]:
```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import plotly.graph_objects as go
from numpy import linalg as LA
from collections import defaultdict
import numpy as np
import random
```

Impresion del csv. Donde se ven los aeropuertos de origen y destino de cada vuelo.

[2]:
```
world = pd.read_csv("World.csv")
```

[3]:
```
world = world.drop(['ConexionAeropuertos'], axis=1)
world
```

[3]:

	Origen	Destino
0	Papua New Guinea	Australia
1	Papua New Guinea	Philippines
2	Papua New Guinea	Indonesia
3	Papua New Guinea	Solomon Islands
4	Papua New Guinea	Hong Kong
…	…	…
2847	Lithuania	Georgia
2848	Armenia	Georgia
2849	Eritrea	Yemen
2850	Yemen	Djibouti
2851	Solomon Islands	Nauru

[2852 rows x 2 columns]

Impresion del .csv Donde se ven los datos de todos los paises del mundo.

Ejercicio 1

Determinar:

a. El diámetro de la red.

b. El grado promedio de la red.

c. El coeficiente de clustering promedio de la red.

[4]:
```
## Genero el grafo que va a representar la red. ##
grafo = nx.from_pandas_edgelist(world, 'Origen', 'Destino')
```

[5]:
```
### Ejercicio 1A: Determinacion del diametro de la red ###

## El diametro de una red se puede definir como largo máximo de todos los␣
 ↪caminos mínimos ##

print("El diametro de la red es", nx.diameter(grafo))
```

```
El diametro de la red es 5
```

[6]:
```
### Ejercicio 1B: Determinacion del grado promedio de la red ###

"""
algoritmo sacado de:
https://github.com/mbuchwald
"""

def grado_promedio(grafo):
    total = 0
    for v in grafo:
        total += len(list(grafo.neighbors(v)))
    return total / len(grafo)

grado_promedio_grafo = round(grado_promedio(grafo), 2)
print("El grado promedio de la red es", grado_promedio_grafo)
```

```
El grado promedio de la red es 24.91
```

[7]:
```
### Ejercicio 1B: Determinacion del coeficiente de clustering promedio de la␣
 ↪red. ###

## Que se define en promedio como La probabilidad de que dos vértices␣
 ↪adyacentes
## del grafo sean adyacentes entre sí.

valores_clustering = nx.clustering(grafo).values()
coef_clustering_promedio = np.array(list(valores_clustering)).mean()
```

```
print("El coeficiente de clustering promedio de la red es",␣
 ↪round(coef_clustering_promedio, 2))
```

```
El coeficiente de clustering promedio de la red es 0.66
```

Ejercicio 2

Indicar si existe algún tipo de Homofilia y qué tipo de homofilia es. Si no hay homofilia por ningún criterio, explicar. Justificar detalladamente.

Primero, vamos a modificar el dataframe para buscar la homofilia por zona geografica.

[8]:
```
COTW = pd.read_csv("CountriesOfTheWorld.csv")
COTW
```

[8]:

	Country	Region	Population	\
0	Afghanistan	ASIA (EX. NEAR EAST)	31056997	
1	Albania	EASTERN EUROPE	3581655	
2	Algeria	NORTHERN AFRICA	32930091	
3	American Samoa	OCEANIA	57794	
4	Andorra	WESTERN EUROPE	71201	
..	…	…	…	
222	West Bank	NEAR EAST	2460492	
223	Western Sahara	NORTHERN AFRICA	273008	
224	Yemen	NEAR EAST	21456188	
225	Zambia	SUB-SAHARAN AFRICA	11502010	
226	Zimbabwe	SUB-SAHARAN AFRICA	12236805	

	Area (sq. mi.)	Pop. Density (per sq. mi.)	Coastline (coast/area ratio)	\
0	647500	48,0	0,00	
1	28748	124,6	1,26	
2	2381740	13,8	0,04	
3	199	290,4	58,29	
4	468	152,1	0,00	
..	…	…	…	
222	5860	419,9	0,00	
223	266000	1,0	0,42	
224	527970	40,6	0,36	
225	752614	15,3	0,00	
226	390580	31,3	0,00	

	Net migration	Infant mortality (per 1000 births)	GDP ($ per capita)	\
0	23,06	163,07	700.0	
1	-4,93	21,52	4500.0	
2	-0,39	31	6000.0	
3	-20,71	9,27	8000.0	
4	6,6	4,05	19000.0	
..	…	…	…	
222	2,98	19,62	800.0	

```
223           NaN                                NaN                NaN
224             0                               61,5              800.0
225             0                              88,29              800.0
226             0                              67,69             1900.0

    Literacy (%) Phones (per 1000) Arable (%) Crops (%) Other (%) Climate  \
0          36,0               3,2      12,13      0,22     87,65       1
1          86,5              71,2      21,09      4,42     74,49       3
2          70,0              78,1       3,22      0,25     96,53       1
3          97,0             259,5         10        15        75       2
4         100,0             497,2       2,22         0     97,78       3
..          …                 …          …         …        …         …
222         NaN             145,2       16,9     18,97     64,13       3
223         NaN               NaN       0,02         0     99,98       1
224        50,2              37,2       2,78      0,24     96,98       1
225        80,6               8,2       7,08      0,03      92,9       2
226        90,7              26,8       8,32      0,34     91,34       2

    Birthrate Deathrate Agriculture Industry Service
0        46,6     20,34        0,38     0,24    0,38
1       15,11      5,22       0,232    0,188   0,579
2       17,14      4,61       0,101      0,6   0,298
3       22,46      3,27         NaN      NaN     NaN
4        8,71      6,25         NaN      NaN     NaN
..        …         …           …        …       …
222     31,67      3,92        0,09     0,28    0,63
223       NaN       NaN         NaN      NaN     0,4
224     42,89       8,3       0,135    0,472   0,393
225        41     19,93        0,22     0,29   0,489
226     28,01     21,84       0,179    0,243   0,579

[227 rows x 20 columns]
```

[9]:
```
COTW_new = COTW.iloc[:,[0,1]]
COTW_new = COTW_new.rename(columns={'Country': 'Origen'})

array_world = world.to_numpy()

for i in range(0, len(array_world)):
    array_world[i][0] = array_world[i][0] + ' '
    array_world[i][1] = array_world[i][1] + ' '

new_world = pd.DataFrame(array_world, columns = ["Origen", "Destino"])

merged = new_world.merge(COTW_new, left_on='Origen', right_on='Origen',␣
 ↪how='outer')
merged = merged.dropna()
```

```
merged
```

[9]:

	Origen	Destino	Region
0	Papua New Guinea	Australia	OCEANIA
1	Papua New Guinea	Philippines	OCEANIA
2	Papua New Guinea	Indonesia	OCEANIA
3	Papua New Guinea	Solomon Islands	OCEANIA
4	Papua New Guinea	Hong Kong	OCEANIA
…	…	…	…
2847	Lithuania	Georgia	BALTICS
2848	Armenia	Georgia	C.W. OF IND. STATES
2849	Eritrea	Yemen	SUB-SAHARAN AFRICA
2850	Yemen	Djibouti	NEAR EAST
2851	Solomon Islands	Nauru	OCEANIA

[2740 rows x 3 columns]

[10]:
```
grafo_mergeado = nx.from_pandas_edgelist(merged, 'Origen', 'Destino')

columna_region_origen = merged['Region']

print('Las regiones de origen y su cantidad son: ')
columna_region_origen.to_frame(name='Region').value_counts()
```

```
Las regiones de origen y su cantidad son:
```

[10]:
```
Region
WESTERN EUROPE                  915
SUB-SAHARAN AFRICA              458
NEAR EAST                       293
LATIN AMER. & CARIB             248
ASIA (EX. NEAR EAST)            218
EASTERN EUROPE                  182
NORTHERN AFRICA                 137
OCEANIA                          96
C.W. OF IND. STATES              95
NORTHERN AMERICA                 75
BALTICS                          23
dtype: int64
```

[11]:
```
merged_vectorizado = merged.to_numpy()

mapper = {}
for i in range(len(merged_vectorizado)):
    mapper[merged_vectorizado[i][0]] = merged_vectorizado[i][2]

def mapper_func(k):
```

```
    try:
        return mapper[k]
    except:
        'None'
```

[12]:
```
def contar_aristas(grafo):
    contador = 0
    for v in grafo:
        contador += len(list(grafo.neighbors(v)))
    return contador if nx.is_directed(grafo) else contador // 2

def proporcion_cruzan_campo(grafo, mapper=None):
    """
    algoritmo sacado de https://github.com/mbuchwald
    """
    aristas_totales = contar_aristas(grafo)
    cruzan_bloque = 0
    visitados = set()
    mapper = mapper if mapper is not None else (lambda k: grafo[k]["type"])

    for v in grafo:
        for w in grafo.neighbors(v):
            if not nx.is_directed(grafo) and w in visitados:
                continue
            if mapper(v) != mapper(w):
                cruzan_bloque += 1
        visitados.add(v)
    return cruzan_bloque / aristas_totales

proporcion_real = round(proporcion_cruzan_campo(grafo_mergeado, mapper_func), 2)

# Aca calcule la 'proporcion de cruce de campo', que es la homofilia global␣
 ↪real, ya que es la
# proporcion de aristas que cruzan grupos sobre el total de aristas disponibles
```

[13]:
```
def proporcion_por_tipo(grafo, mapper=None):
    """
    algoritmo sacado de https://github.com/mbuchwald
    """
    mapper = mapper if mapper is not None else (lambda k: grafo[k]["type"])
    cantidades = {}
    for v in grafo:
        cantidades[mapper(v)] = cantidades.get(mapper(v), 0) + 1
    props = {}
    for c in cantidades:
        props[c] = cantidades[c] / len(grafo)
    return props
```

```
proporciones = proporcion_por_tipo(grafo_mergeado, mapper_func)

proporcion_teorica = 0

for clave, valor in proporciones.items():
    proporcion_teorica = proporcion_teorica + (proporciones[clave] * (1 -␣
 ↪proporciones[clave]))

# Aca calcule la homofilia global teorica, que es la sumatoria de todas
# las probabilidades de agarrar un nodo y que pertenezca a un cierto grupo
```

[14]:
```
proporcion_real_vs_teorico = round(proporcion_real * 100 / proporcion_teorica,␣
 ↪2)
```

[15]:
```
def proporcion_cruzan_campo_de_tipo(grafo, tipo, mapper=None):
    """
    algoritmo sacado de https://github.com/mbuchwald
    """
    cruzan_bloque = 0
    visitados = set()
    aristas = 0
    mapper = mapper if mapper is not None else (lambda k: grafo[k]["type"])

    for v in grafo:
        if mapper(v) != tipo:
            continue
        for w in grafo.neighbors(v):
            aristas += 1
            if mapper(v) != mapper(w):
                cruzan_bloque += 1
        visitados.add(v)
    return cruzan_bloque / aristas

proporciones_cruce_campo = {}

for clave, valor in proporciones.items():
    if clave != None:
        proporciones_cruce_campo[clave] =␣
 ↪round(proporcion_cruzan_campo_de_tipo(grafo_mergeado, \

                                                                        ␣
 ↪clave, mapper_func),2)
```

Ahora, ya que se calcularon todos los valores los mostrare aqui abajo

[16]:
```
print("Homofilia global real:", proporcion_real)
print("Homofilia global teorica:", round(proporcion_teorica,2))
```

```
print("Proporcion de homofilia real vs teorica", proporcion_real_vs_teorico)
print("-------------------------------------------")
print("Homofilia por grupos:")

for clave, valor in proporciones_cruce_campo.items():
    print(valor, '-', clave)
```

```
Homofilia global real: 0.67
Homofilia global teorica: 0.88
Proporcion de homofilia real vs teorica 76.35
-------------------------------------------
Homofilia por grupos:
0.55 - OCEANIA
0.56 - ASIA (EX. NEAR EAST)
0.99 - NORTHERN AMERICA
0.72 - WESTERN EUROPE
0.79 - EASTERN EUROPE
0.93 - BALTICS
0.8 - NEAR EAST
0.38 - LATIN AMER. & CARIB
0.9 - NORTHERN AFRICA
0.46 - SUB-SAHARAN AFRICA
0.8 - C.W. OF IND. STATES
```

Estos resultados pueden interpretarse de muchas formas distintas. Por ejemplo que tiene mucho sentido que haya poca homofilia en el norte de america, primero y principal porque los paises de esa region son dos (Canada y Estados Unidos) y solo habria homofilia en los vuelos internos. Aparte al ser paises muy desarrollados y con lugares geograficos estrategicos (punto entre el pacifico y el atlantico) es muy comun que haya escalas en esos lugares y que permitan conexiones con paises de otras partes del globo, rompiendo asi la homofilia. Tambien tiene mucho sentido que exista bastante homofilia en el africa subsahariana y latinoamerica, ya que son zonas no tan desarrolladas (en algunos casos de paises) y no tiene lugares geograficos estrategicos, realizandose asi vuelos mas que nada zonales y muy pocos de larga distancia. Los paises de ocenia tambien podrian ser otro caso claro de homofilia por su lejania a los demas paises del globo. Luego en la comparacion de la homofilia real con la teorica, se obtiene que la cercania entre la teorica y la real es mayor al 75%, lo cual a mi criterio es un buen valor y da a entender que los calculos han sido realizados de una manera correcta.

Ejercicio 3

Determinar los puentes (globales o locales) en dicha red.

[17]:
```
puentes = nx.local_bridges(grafo)

list(puentes)
```

[17]:
```
[('Papua New Guinea', 'Micronesia', 3),
 ('Fiji', 'Tuvalu', inf),
 ('Micronesia', 'Marshall Islands', 3),
```

```
('United States', 'American Samoa', inf),
('United Kingdom', 'Saint Helena', inf),
('Canada', 'Saint Pierre and Miquelon', inf),
('Antigua and Barbuda', 'Montserrat', inf),
('New Zealand', 'Niue', inf),
('South Africa', 'Lesotho', inf),
('South Africa', 'Swaziland', inf),
('Burma', 'Myanmar', inf)]
```

Los puentes son los que estan impresos arriba. Se puede ver vectores de 3 elementos siendo los primeros dos los nodos que el puente conecta y luego la longitud del camino que los une si este es removido. Los que tienen longitud 'inf' significa que tienen longitud infinita, quiere decir que no se puede llegar de el primer nodo al segundo quitando esa arista, lo cual es la caracteristica de los puentes globales. Dejando el grafo de manera inconexa. Luego, vemos que hay dos que tienen distancia tres, esos puentes se los puede determinar como puentes locales, ya que el grafo sigue siendo conexo y hay un camino alternativo que une los nodos.

Ejercicio 4

a. Determinar un tipo de centralidad que podría ser útil calcular para esta red, justificando.

b. Realizar una representación gráfica de dicha red, considerando la centralidad de los distintos países dada por la métrica del punto a (tamaño de los nodos proporcional a dicha métrica).

a. Para esta red considero que hay dos tipos utiles de centralidades para utilizar. La primera es "centralidad de grado", el cual tiene grafico bastante facil de entender. Se grafican los nodos por tamaño segun cuantas aristas entrantes tenga y tambien en escalas de colores que marcan cuales son los mas importantes. Esta es una buena eleccion ya que los aeropuertos que reciben mas aviones se notaran en el grafico representando que son los mas importantes. La otra centralidad a utilizar es "betweenness centrality" la cual denota como nodos mas imporantes a los que aparecen como intermediario en algún camino mínimo. Lo interesante de este tipo de centralidad es que mostraria que aeropuertos estan en lugares estrategicos para el trafico de los aviones.

[18]:
```
edge_x = []
edge_y = []

pos = nx.spring_layout(grafo, iterations=50, k=2)

for edge in grafo.edges():
    x0, y0 = pos[edge[0]][0], pos[edge[0]][1]
    x1, y1 = pos[edge[1]][0], pos[edge[1]][1]
    edge_x.append(x0)
    edge_x.append(x1)
    edge_x.append(None)
    edge_y.append(y0)
    edge_y.append(y1)
    edge_y.append(None)
```

```
edge_trace = go.Scatter(
    x=edge_x, y=edge_y,
    line=dict(width=0.5, color='#CFCFCF'),
    hoverinfo='none',
    mode='lines')

node_x = []
node_y = []
node_label = []
for node in grafo.nodes():
    x, y = pos[node][0], pos[node][1]
    node_x.append(x)
    node_y.append(y)
    node_label.append(node)

node_adjacencies = []
node_size = []
for node, adjacencies in enumerate(grafo.adjacency()):
    node_adjacencies.append(len(adjacencies[1]))
for i in range(len(node_adjacencies)):
    node_size.append(np.log(node_adjacencies[i])*5)

node_trace = go.Scatter(
    x=node_x, y=node_y,
    mode='markers+text',
    hoverinfo='text',
    text=node_label,
    textposition='bottom center',
     textfont=dict(
        family="sans serif",
        size=6,
        color="Black"
    ),
    marker=dict(
        showscale=True,
        colorscale='Hot',
        reversescale=True,
        color=[],
        size=node_size,
        colorbar=dict(
            thickness=15,
            title='Grado del nodo',
            xanchor='left',
            titleside='right'
        ),
        line_width=2))
```

```
node_trace.marker.color = node_adjacencies

fig = go.Figure(data=[edge_trace, node_trace],
             layout=go.Layout(
                title='<br>Centralidad de grado',
                titlefont_size=16,
                showlegend=False,
                hovermode='closest',
                margin=dict(b=20,l=5,r=5,t=40),
                annotations=[ dict(
                    showarrow=False,
                    xref="paper", yref="paper",
                    x=0.005, y=-0.002 ) ],
                xaxis=dict(showgrid=False, zeroline=False,␣
 ↪showticklabels=False),
                yaxis=dict(showgrid=False, zeroline=False,␣
 ↪showticklabels=False))
                )
fig.show()
```

Para una mejor experiencia con los graficos se recomienda verlos desde el notebook. Ya que es interactivo.

[19]:
```
edge_x = []
edge_y = []

pos = nx.spring_layout(grafo, iterations=50, k=2)

for edge in grafo.edges():
    x0, y0 = pos[edge[0]][0], pos[edge[0]][1]
    x1, y1 = pos[edge[1]][0], pos[edge[1]][1]
    edge_x.append(x0)
    edge_x.append(x1)
    edge_x.append(None)
    edge_y.append(y0)
    edge_y.append(y1)
    edge_y.append(None)

edge_trace = go.Scatter(
    x=edge_x, y=edge_y,
    line=dict(width=0.5, color='#CFCFCF'),
    hoverinfo='none',
    mode='lines')

node_x = []
node_y = []
node_label = []
```

```
for node in grafo.nodes():
    x, y = pos[node][0], pos[node][1]
    node_x.append(x)
    node_y.append(y)
    node_label.append(node)

node_adjacencies = []
node_size = []
for node, adjacencies in enumerate(grafo.adjacency()):
    node_adjacencies.append(len(adjacencies[1]))

betweenness_centrality = nx.betweenness_centrality(grafo)
for e in betweenness_centrality:
    node_size.append(betweenness_centrality[e]*100+10)

node_trace = go.Scatter(
    x=node_x, y=node_y,
    mode='markers+text',
    hoverinfo='text',
    text=node_label,
    textposition='bottom center',
     textfont=dict(
        family="sans serif",
        size=6,
        color="Black"
    ),
    marker=dict(
        showscale=True,
        colorscale='Hot',
        reversescale=True,
        color=[],
        size=node_size,
        colorbar=dict(
            thickness=15,
            title='(centralidad del nodo * 100) + 10',
            xanchor='left',
            titleside='right'
        ),
        line_width=2))

node_trace.marker.color = node_adjacencies

fig = go.Figure(data=[edge_trace, node_trace],
             layout=go.Layout(
                title='<br>Betweenness centrality',
                titlefont_size=16,
                showlegend=False,
```

```
                hovermode='closest',
                margin=dict(b=20,l=5,r=5,t=40),
                annotations=[ dict(
                    showarrow=False,
                    xref="paper", yref="paper",
                    x=0.005, y=-0.002 ) ],
                xaxis=dict(showgrid=False, zeroline=False,␣
 ↪showticklabels=False),
                yaxis=dict(showgrid=False, zeroline=False,␣
 ↪showticklabels=False))
                )
fig.show()
```

Ejercicio 5

a. Obtener una simulación de un modelado de Erdös-Rényi que corresponda a los parámetros de esta red.

b. Obtener una simulación de un modelado de Preferential Attachment (ley de potencias) que corresponda a los parámetros de esta red.

c. Obtener una representación de anonymous walks tanto de la red original como para las dos simuladas en los puntos a y b. Determinar por distancia coseno cuál sería la simulación más afín.

[20]:
```
# Punto a.
cantidad_nodos = len(grafo)
erdos_renyi = nx.erdos_renyi_graph(cantidad_nodos, coef_clustering_promedio)

print("ERDOS RENYI")
print("El grado promedio de la red es", round(grado_promedio(erdos_renyi), 2))
print("El diametro de la red es", nx.diameter(erdos_renyi))
```

```
ERDOS RENYI
El grado promedio de la red es 150.12
El diametro de la red es 2
```

Los valores obtenidos con la simulacion de erdos renyi comparados con los de nuestro grafo son totalmente distintos. En el caso del grado promedio con la simulacion aleatoria el resultado fue 150, un resultado totalmente distinto al de nuestro grafo inicial que es 24. Lo mismo con el diametro de la red, que en el caso de erdos renyi el valor obtenido es menos de la mitad que el conseguido inicialmente. Se puede concluir que en este caso el modelado no es tan preciso.

[21]:
```
# Punto b.

"""
algoritmos sacados de:
github.com/mbuchwald
"""
```

```python
def alfa_preferential_attachment(grafo, x_m):
    sumatoria = 0
    for v in grafo:
        cant_ady = len(list(grafo.neighbors(v)))
        if cant_ady >= x_m:
            sumatoria += np.log(cant_ady / x_m)
    return 1 + len(grafo) / sumatoria

def elegir_preferntial(grafo, banned):
    grados_entrada = [0] * len(grafo)
    total = 0
    for v in range(len(grafo)):
        for w in grafo.neighbors(v):
            if w in banned:
                continue
            grados_entrada[w] += 1
            total += 1
    if total == 0:
        return None

    aleat = random.uniform(0, total)
    sumando = 0
    for i in range(len(grafo)):
        sumando += grados_entrada[i]
        if sumando > aleat:
            return i

def preferential_attachment(dirigido, alfa, cant, k):
    p = 1 - (1/(alfa - 1))
    valores = list(range(cant))
    g = nx.DiGraph() if dirigido else nx.Graph()
    g.add_nodes_from(valores)
    for v in range(cant):
        banned = set([v])
        for i in range(int(k)):
            preferential = random.uniform(0, 1) < p
            ya_agregado = False
            if preferential and v > 0:
                w = elegir_preferntial(g, banned)
                if w is not None:
                    banned.add(w)
                    g.add_edge(v, w)
                    ya_agregado = False
            if not ya_agregado:
                w = random.choice(list(set(range(cant)) - set([v])))
                g.add_edge(v, w)
```

```
    return g
```

[22]:
```
grafo_pa = preferential_attachment(False, alfa_preferential_attachment(grafo,␣
↪10), \
                                   len(grafo), grado_promedio_grafo)

print("PREFERENTIAL ATTACHMENT")
print("El grado promedio de la red es", round(grado_promedio(grafo_pa), 2))
print("El diametro de la red es", nx.diameter(grafo_pa))
```

```
PREFERENTIAL ATTACHMENT
El grado promedio de la red es 54.73
El diametro de la red es 2
```

Los valores obtenidos con la simulacion de preferential attachment comparados con los de nuestro grafo son tambien distintos, al igual que en el caso de erdos renyi. En el caso del grado promedio con la simulacion aleatoria el resultado fue 54, un resultado totalmente distinto al de nuestro grafo inicial pero un poco mas acertado que el algortimo aleatorio anteriormente analizado. En cuanto al diametro de la red, el resultado esta igual de alejado que erdos renyi. La diferencia es de 2 versus 5.

[23]:
```
# Punto c.

"""
algoritmos sacados de:
github.com/mbuchwald
"""

from math import log, e, ceil
import numpy as np
import random
from numpy import linalg as LA

DIFFERENT_WALKS = { # incluye bucles
    2: 2,
    3: 5,
    4: 15,
    5: 52,
    6: 203,
    7: 877,
    8: 4140,
    9: 21147,
    10: 115975,
    11: 678570,
    12: 4213597
}
```

```
def ln(num):
    return log(num, e)


def _cant_annonymous_walks(length, error=0.1, delta=0.01):
    nu = DIFFERENT_WALKS[length]
    return ceil((ln(2**nu - 2) - ln(delta)) * (2 / (error**2)))


def _camino_a_clave(camino):
    return "-".join(map(lambda v: str(v), camino))


def _annon_enum_rec(pasos_restantes, mapeo, camino=[], vs_en_camino=0,␣
 ↪admite_bucles=False):
    if pasos_restantes == 0:
        mapeo[_camino_a_clave(camino)] = len(mapeo)
        return
    nuevo = vs_en_camino + 1
    ultimo = camino[-1] if len(camino) > 0 else None
    for i in range(1, nuevo + 1):
        if i == ultimo and not admite_bucles:
            continue
        camino.append(i)
        vs_en_este_camino = vs_en_camino + (1 if i == nuevo else 0)
        _annon_enum_rec(pasos_restantes - 1, mapeo, camino, vs_en_este_camino,␣
 ↪admite_bucles)
        camino.pop()


def _enumerar_anonymous_walks(length, admite_bucles=False):
    mapeo = {}
    _annon_enum_rec(length, mapeo, admite_bucles=admite_bucles)
    return mapeo


def _random_walk(grafo, length):
    v = random.choice(list(grafo.nodes))
    camino = [v]
    while len(camino) < length:
        v = random.choice(list(grafo.neighbors(v)))
        camino.append(v)
    return camino


def _anonymize_walk(camino):
    translate = {}
```

```
    camino_trans = []
    for v in camino:
        if v not in translate:
            translate[v] = len(translate) + 1
        camino_trans.append(translate[v])
    return camino_trans


def anoymous_walks(grafo, length, admite_bucles=False):
    cantidad = _cant_annonymous_walks(length)
    mapeo = _enumerar_anonymous_walks(length, admite_bucles)
    contadores = [0] * len(mapeo)
    for i in range(cantidad):
        camino = _random_walk(grafo, length)
        contadores[mapeo[_camino_a_clave(_anonymize_walk(camino))]] += 1

    vector = np.array(contadores)
    return list(mapeo.keys()), vector / LA.norm(vector)
```

[24]:
```
an_walks_grafo_inicial = anoymous_walks(grafo, 7)[-1]
an_walks_erdos_renyi   = anoymous_walks(erdos_renyi, 7)[-1]
an_walks_pref_attach   = anoymous_walks(grafo_pa, 7)[-1]

def distancia_coseno(a, b):
    return 1 - np.inner(a, b) / (LA.norm(a) * LA.norm(b))

distancia_coseno_ER = round(distancia_coseno(an_walks_grafo_inicial,␣
 ↪an_walks_erdos_renyi),4)
distancia_coseno_PA = round(distancia_coseno(an_walks_grafo_inicial,␣
 ↪an_walks_pref_attach),4)

print("La distancia coseno entre el grafo inicial y el grafo obtenido por Erdos␣
 ↪Renyi es", \
     distancia_coseno_ER)
print("La distancia coseno entre el grafo inicial y el grafo obtenido por Pref␣
 ↪Attachtment es "\
      ,distancia_coseno_PA)

if distancia_coseno_PA < distancia_coseno_ER:
    print("La simulacion mas afin es Preferential Attachment")
else:
    print("La simulacion mas afin es Erdos Renyi")
```

```
La distancia coseno entre el grafo inicial y el grafo obtenido por Erdos Renyi
es 0.0042
La distancia coseno entre el grafo inicial y el grafo obtenido por Pref
Attachtment es  0.0022
```

La simulacion mas afin es Preferential Attachment