import networkx as nx

import numpy as np

from networkx.algorithms import bipartite

import matplotlib.pyplot as plt

#characters.txt contains a list of nodes referring to each character

#comics.txt contains a list of nodes referring to each comic book

#out.txt coints edge-list of characters and comics

characters = np.loadtxt('characters.txt',dtype=int)

comics = np.loadtxt('comics.txt',dtype=int)

B = nx.Graph()

B.add\_nodes\_from(characters,bipartite=0)

B.add\_nodes\_from(comics,bipartite=1)

h = nx.read\_edgelist('out.txt',create\_using=nx.Graph(),nodetype=int)

B.add\_edges\_from(h.edges)

#creating unipartite network out of bipartite network

G = bipartite.weighted\_projected\_graph(B,characters)

#getting giant component graph

cc = list(max(nx.connected\_components(G),key=len))

file0 = open("connected.txt","w")

for i in cc:

file0.write("%d\n"%i)

file0.close()

#Computing degree centrality

degCent = nx.degree\_centrality(G)

#storing in a text file

with open('degCent.txt', 'w') as f:

print(degCent, file=f)

#sorting degree centrality in descending order

degCent\_sorted=dict(sorted(degCent.items(), key=lambda item: item[1],reverse=True))

#storing in a text file

with open('degCent\_sorted.txt', 'w') as f:

print(degCent\_sorted, file=f)

#getting top 100 nodes

print("The top 100 nodes of the network sorted on the basis of their degree centrality in descending order are:")

N\_top=100

keys\_deg\_top=list(degCent\_sorted)[0:N\_top]

d = {}

with open("dict.txt",encoding="utf8") as file:

for line in file:

line = line.split(':')

if not line:

continue

d[line[0]] = line[1:]

top\_nodes = keys\_deg\_top

for i in range(len(top\_nodes)):

print(top\_nodes[i],"\t:\t",d[str(top\_nodes[i])][0])

#getting lowest 10 nodes

print("The lowest 10 nodes of the network sorted on the basis of their degree centrality in descending order are:")

N\_low=10

keys\_deg\_low=list(degCent\_sorted)[6476:6486]

d = {}

with open("dict.txt",encoding="utf8") as file:

for line in file:

line = line.split(':')

if not line:

continue

d[line[0]] = line[1:]

low\_nodes = keys\_deg\_low

for i in range(len(low\_nodes)):

print(low\_nodes[i],"\t:\t",d[str(low\_nodes[i])][0])

#SMALL WORLD

#making a giant component graph

connected = np.loadtxt('connected.txt',dtype=int)

GC = bipartite.weighted\_projected\_graph(B,connected)

#Calculating clustering coefficient for GC

average\_clustering\_coefficient\_GC = nx.average\_clustering(GC)

print("Average\_Clustering\_Coefficient\_GC:",average\_clustering\_coefficient\_GC)

# Calculate the average shortest path length for GC

average\_shortest\_path\_length\_GC = nx.average\_shortest\_path\_length(GC)

print("Average Shortest Path Length for GC:", average\_shortest\_path\_length\_GC)

#average-degree calculation for giant component

deg = 0

kavg = 0

maximum = 0

loc = 0

for i in GC.nodes():

deg+=GC.degree[i]

if(GC.degree[i]>maximum):

maximum = GC.degree[i]

loc = i

kavg = deg/len(GC.degree)

print("Average Degree :",kavg)

#p=kavg/n;where n=nodes

n=6449

p=kavg/(n)

print("The fixed value of p is:",p)

#Watt-Strogatz Model

average\_clustering\_coefficient=0

average\_shortest\_path\_length=0

j=0

for i in range(25):

w=nx.watts\_strogatz\_graph(n, round(kavg) , p, seed=None)

print(w)

#Calculating clustering coefficient for w

average\_clustering\_coefficient\_w = nx.average\_clustering(w)

print("Average\_Clustering\_Coefficient\_w:",average\_clustering\_coefficient\_w)

average\_clustering\_coefficient=average\_clustering\_coefficient+average\_clustering\_coefficient\_w

# Calculate the average shortest path length for w

average\_shortest\_path\_length\_w = nx.average\_shortest\_path\_length(w)

print("Average Shortest Path Length for w:", average\_shortest\_path\_length\_w)

average\_shortest\_path\_length=average\_shortest\_path\_length+ average\_shortest\_path\_length\_w

j=j+1

print("Average\_Clustering\_Coefficient\_w for ",j," random graphs",average\_clustering\_coefficient/j)

print("Average\_shortest\_path\_length\_w for ",j," random graphs",average\_shortest\_path\_length/j)

#getting the diameter of the giant component.

print(nx.diameter(GC))