Lesson 30, Quiz: Gamer’s Network:

# --------------------------- #

# Intro to CS Final Project #

# Gaming Social Network #

# --------------------------- #

#

# Background

# ==========

# You and your friend have decided to start a company that hosts a gaming

# social network site. Your friend will handle the website creation (they know

# what they are doing, having taken our web development class). However, it is

# up to you to create a data structure that manages the game-network information

# and to define several procedures that operate on the network.

#

# In a website, the data is stored in a database. In our case, however, all the

# information comes in a big string of text. Each pair of sentences in the text

# is formatted as follows:

#

# <user> is connected to <user1>, ..., <userM>.<user> likes to play <game1>, ..., <gameN>.

#

# For example:

#

# John is connected to Bryant, Debra, Walter.John likes to play The Movie: The Game, The Legend of Corgi, Dinosaur Diner.

#

# Note that each sentence will be separated from the next by only a period. There will

# not be whitespace or new lines between sentences.

#

# Your friend records the information in that string based on user activity on

# the website and gives it to you to manage. You can think of every pair of

# sentences as defining a user's profile.

#

# Consider the data structures that we have used in class - lists, dictionaries,

# and combinations of the two (e.g. lists of dictionaries). Pick one that

# will allow you to manage the data above and implement the procedures below.

#

# You may assume that <user> is a unique identifier for a user. For example, there

# can be at most one 'John' in the network. Furthermore, connections are not

# symmetric - if 'Bob' is connected to 'Alice', it does not mean that 'Alice' is

# connected to 'Bob'.

#

# Project Description

# ====================

# Your task is to complete the procedures according to the specifications below

# as well as to implement a Make-Your-Own procedure (MYOP). You are encouraged

# to define any additional helper procedures that can assist you in accomplishing

# a task. You are encouraged to test your code by using print statements and the

# Test Run button.

# -----------------------------------------------------------------------------

# Example string input. Use it to test your code.

example\_input = "John is connected to Bryant, Debra, Walter.\

John likes to play The Movie: The Game, The Legend of Corgi, Dinosaur Diner.\

Bryant is connected to Olive, Ollie, Freda, Mercedes.\

Bryant likes to play City Comptroller: The Fiscal Dilemma, Super Mushroom Man.\

Mercedes is connected to Walter, Robin, Bryant.\

Mercedes likes to play The Legend of Corgi, Pirates in Java Island, Seahorse Adventures.\

Olive is connected to John, Ollie.\

Olive likes to play The Legend of Corgi, Starfleet Commander.\

Debra is connected to Walter, Levi, Jennie, Robin.\

Debra likes to play Seven Schemers, Pirates in Java Island, Dwarves and Swords.\

Walter is connected to John, Levi, Bryant.\

Walter likes to play Seahorse Adventures, Ninja Hamsters, Super Mushroom Man.\

Levi is connected to Ollie, John, Walter.\

Levi likes to play The Legend of Corgi, Seven Schemers, City Comptroller: The Fiscal Dilemma.\

Ollie is connected to Mercedes, Freda, Bryant.\

Ollie likes to play Call of Arms, Dwarves and Swords, The Movie: The Game.\

Jennie is connected to Levi, John, Freda, Robin.\

Jennie likes to play Super Mushroom Man, Dinosaur Diner, Call of Arms.\

Robin is connected to Ollie.\

Robin likes to play Call of Arms, Dwarves and Swords.\

Freda is connected to Olive, John, Debra.\

Freda likes to play Starfleet Commander, Ninja Hamsters, Seahorse Adventures."

# -----------------------------------------------------------------------------

# create\_data\_structure(string\_input):

# Parses a block of text (such as the one above) and stores relevant

# information into a data structure. You are free to choose and design any

# data structure you would like to use to manage the information.

#

# Arguments:

# string\_input: block of text containing the network information

#

# You may assume that for all the test cases we will use, you will be given the

# connections and games liked for all users listed on the right-hand side of an

# 'is connected to' statement. For example, we will not use the string

# "A is connected to B.A likes to play X, Y, Z.C is connected to A.C likes to play X."

# as a test case for create\_data\_structure because the string does not

# list B's connections or liked games.

#

# The procedure should be able to handle an empty string (the string '') as input, in

# which case it should return a network with no users.

#

# Return:

# The newly created network data structure

def cleaned\_list(original\_sentence, remove\_string):

reduced\_string = original\_sentence.replace(remove\_string, '')

list\_with\_spaces = reduced\_string.split(',')

clean\_list = list()

for item in list\_with\_spaces:

item = item.strip()

clean\_list.append(item)

return clean\_list

def create\_data\_structure(string\_input):

network = dict()

if string\_input:

# split\_input = ['John is connected to Bryant, Debra, Walter', 'John likes to play The Movie: The Game, The

# Legend of Corgi, Dinosaur Diner', ...]

split\_input = string\_input.split('.')

for sentence in split\_input:

if sentence:

first\_name = sentence.split(' ')[0] # LOOPS = John, John, Bryant, Bryant, Mercedes, ...

# Creates a space in the network for first\_name if it is not currently in the list.

if first\_name not in network:

network[first\_name] = dict()

if sentence.find(' is connected to ') > -1:

remove\_string = first\_name + ' is connected to '

network[first\_name]['connections'] = cleaned\_list(sentence, remove\_string)

else:

remove\_string = first\_name + ' likes to play '

network[first\_name]['games'] = cleaned\_list(sentence, remove\_string)

return network

# PASS form dict(dict('connections': [], 'games': []))

# print(create\_data\_structure(example\_input))

#

# {'John': {'connections': ['Bryant', 'Debra', 'Walter'], 'games': ['The Movie: The Game', 'The Legend of Corgi',

# 'Dinosaur Diner']}, 'Bryant': {'connections': ['Olive', 'Ollie', 'Freda', 'Mercedes'], 'games': ['City Comptroller:

# The Fiscal Dilemma', 'Super Mushroom Man']}, 'Mercedes': {'connections': ['Walter', 'Robin', 'Bryant'], 'games': ['The

# Legend of Corgi', 'Pirates in Java Island', 'Seahorse Adventures']}, 'Olive': {'connections': ['John', 'Ollie'],

# 'games': ['The Legend of Corgi', 'Starfleet Commander']}, 'Debra': {'connections': ['Walter', 'Levi', 'Jennie',

# 'Robin'], 'games': ['Seven Schemers', 'Pirates in Java Island', 'Dwarves and Swords']}, 'Walter': {'connections':

# ['John', 'Levi', 'Bryant'], 'games': ['Seahorse Adventures', 'Ninja Hamsters', 'Super Mushroom Man']}, 'Levi':

# {'connections': ['Ollie', 'John', 'Walter'], 'games': ['The Legend of Corgi', 'Seven Schemers', 'City Comptroller:

# The Fiscal Dilemma']}, 'Ollie': {'connections': ['Mercedes', 'Freda', 'Bryant'], 'games': ['Call of Arms', 'Dwarves

# and Swords', 'The Movie: The Game']}, 'Jennie': {'connections': ['Levi', 'John', 'Freda', 'Robin'], 'games': ['Super

# Mushroom Man', 'Dinosaur Diner', 'Call of Arms']}, 'Robin': {'connections': ['Ollie'], 'games': ['Call of Arms',

# 'Dwarves and Swords']}, 'Freda': {'connections': ['Olive', 'John', 'Debra'], 'games': ['Starfleet Commander', 'Ninja

# Hamsters', 'Seahorse Adventures']}, '': {'games': ['']}}

# ----------------------------------------------------------------------------- #

# Note that the first argument to all procedures below is 'network' This is the #

# data structure that you created with your create\_data\_structure procedure, #

# though it may be modified as you add new users or new connections. Each #

# procedure below will then modify or extract information from 'network' #

# ----------------------------------------------------------------------------- #

# -----------------------------------------------------------------------------

# get\_connections(network, user):

# Returns a list of all the connections that user has

#

# Arguments:

# network: the gamer network data structure

# user: a string containing the name of the user

#

# Return:

# A list of all connections the user has.

# - If the user has no connections, return an empty list.

# - If the user is not in network, return None.

def get\_connections(network, user):

if user not in network:

return None

return network[user]['connections']

# PASS returns list of connections

# test\_network = create\_data\_structure(example\_input)

# print(get\_connections(test\_network, 'John')) # ['Bryant', 'Debra', 'Walter']

# -----------------------------------------------------------------------------

# get\_games\_liked(network, user):

# Returns a list of all the games a user likes

#

# Arguments:

# network: the gamer network data structure

# user: a string containing the name of the user

#

# Return:

# A list of all games the user likes.

# - If the user likes no games, return an empty list.

# - If the user is not in network, return None.

def get\_games\_liked(network, user):

if user not in network:

return None

return network[user]['games']

# PASS returns list of games

# test\_network = create\_data\_structure(example\_input)

# print(get\_games\_liked(test\_network, 'John')) # ['The Movie: The Game', 'The Legend of Corgi', 'Dinosaur Diner']

# -----------------------------------------------------------------------------

# add\_connection(network, user\_A, user\_B):

# Adds a connection from user\_A to user\_B. Make sure to check that both users

# exist in network.

#

# Arguments:

# network: the gamer network data structure

# user\_A: a string with the name of the user the connection is from

# user\_B: a string with the name of the user the connection is to

#

# Return:

# The updated network with the new connection added.

# - If a connection already exists from user\_A to user\_B, return network unchanged.

# - If user\_A or user\_B is not in network, return False.

def add\_new\_empty\_user(network, user):

if user not in network:

network[user] = dict()

network[user]['connections'] = list()

network[user]['games'] = list()

return network

def add\_connection(network, user\_A, user\_B):

if user\_A not in network or user\_B not in network:

return False

if user\_B not in get\_connections(network, user\_A):

network[user\_A]['connections'].append(user\_B)

return network

# PASS adds Pete to network then to John's connections

# test\_network = create\_data\_structure(example\_input)

# print(add\_connection(test\_network, 'John', 'Robin'))

# {'John': 'connections': ['Bryant', 'Debra', 'Walter', 'Pete'], 'games': ['The Movie: The Game', 'The Legend of Corgi',

# 'Dinosaur Diner']}, ...

# -----------------------------------------------------------------------------

# add\_new\_user(network, user, games):

# Creates a new user profile and adds that user to the network, along with

# any game preferences specified in games. Assume that the user has no

# connections to begin with.

#

# Arguments:

# network: the gamer network data structure

# user: a string containing the name of the user to be added to the network

# games: a list of strings containing the user's favorite games, e.g.:

# ['Ninja Hamsters', 'Super Mushroom Man', 'Dinosaur Diner']

#

# Return:

# The updated network with the new user and game preferences added. The new user

# should have no connections.

# - If the user already exists in network, return network \*UNCHANGED\* (do not change

# the user's game preferences)

def add\_new\_user(network, user, games):

if user not in network:

network[user] = dict()

network[user]['connections'] = list()

network[user]['games'] = games

return network

# PASS added 'Pete': {'connections': [], 'games': ['Kirby Air Ride', 'StarCraft II'] to test\_network

# test\_network = create\_data\_structure(example\_input)

# print(add\_new\_user(test\_network, 'John', ['StarCraft II'])) # Should not alter John

# print(add\_new\_user(test\_network, 'Pete', ['Kirby Air Ride', 'StarCraft II'])) # Add Pete to network and his games

# -----------------------------------------------------------------------------

# get\_secondary\_connections(network, user):

# Finds all the secondary connections (i.e. connections of connections) of a

# given user.

#

# Arguments:

# network: the gamer network data structure

# user: a string containing the name of the user

#

# Return:

# A list containing the secondary connections (connections of connections).

# - If the user is not in the network, return None.

# - If a user has no primary connections to begin with, return an empty list.

#

# NOTE:

# It is OK if a user's list of secondary connections includes the user

# himself/herself. It is also OK if the list contains a user's primary

# connection that is a secondary connection as well.

def get\_secondary\_connections(network, user):

if user not in network:

return None

secondary\_connections = list()

if user in network:

primary\_connections = get\_connections(network, user)

for friend in primary\_connections:

friend\_of\_friend = get\_connections(network, friend)

for person in friend\_of\_friend:

if person not in secondary\_connections:

secondary\_connections.append(person)

return secondary\_connections

# PASS

# test\_network = create\_data\_structure(example\_input)

# print(get\_secondary\_connections(test\_network, 'John')) # ['Olive', 'Ollie', 'Freda', 'Mercedes', 'Walter', 'Levi',

# 'Jennie', 'Robin', 'John', 'Bryant']

# print(get\_secondary\_connections(test\_network, 'Pete')) # []

# -----------------------------------------------------------------------------

# count\_common\_connections(network, user\_A, user\_B):

# Finds the number of people that user\_A and user\_B have in common.

#

# Arguments:

# network: the gamer network data structure

# user\_A: a string containing the name of user\_A

# user\_B: a string containing the name of user\_B

#

# Return:

# The number of connections in common (as an integer).

# - If user\_A or user\_B is not in network, return False.

def count\_common\_connections(network, user\_A, user\_B):

if user\_A not in network or user\_B not in network:

return False

count = 0

user\_a\_connections = get\_connections(network, user\_A)

user\_b\_connections = get\_connections(network, user\_B)

for connection in user\_a\_connections:

if connection in user\_b\_connections:

count += 1

return count

# PASS

# test\_network = create\_data\_structure(example\_input)

# print(count\_common\_connections(test\_network, 'John', 'Bryant')) # 0

# print(count\_common\_connections(test\_network, 'John', 'Mercedes')) # 2

# print(count\_common\_connections(test\_network, 'John', 'Pete')) # 0

# -----------------------------------------------------------------------------

# find\_path\_to\_friend(network, user\_A, user\_B):

# Finds a connections path from user\_A to user\_B. It has to be an existing

# path but it DOES NOT have to be the shortest path.

#

# Arguments:

# network: The network you created with create\_data\_structure.

# user\_A: String holding the starting username ("Abe")

# user\_B: String holding the ending username ("Zed")

#

# Return:

# A list showing the path from user\_A to user\_B.

# - If such a path does not exist, return None.

# - If user\_A or user\_B is not in network, return None.

#

# Sample output:

# >>> print find\_path\_to\_friend(network, "Abe", "Zed")

# >>> ['Abe', 'Gel', 'Sam', 'Zed']

# This implies that Abe is connected with Gel, who is connected with Sam,

# who is connected with Zed.

#

# NOTE:

# You must solve this problem using recursion!

#

# Hints:

# - Be careful how you handle connection loops, for example, A is connected to B.

# B is connected to C. C is connected to B. Make sure your code terminates in

# that case.

# - If you are comfortable with default parameters, you might consider using one

# in this procedure to keep track of nodes already visited in your search. You

# may safely add default parameters since all calls used in the grading script

# will only include the arguments network, user\_A, and user\_B.

def find\_path\_to\_friend(network, user\_A, user\_B):

if user\_A not in network or user\_B not in network:

return None

else:

user\_a\_connections = get\_connections(network, user\_A)

if user\_B in user\_a\_connections:

return [user\_A, user\_B]

else:

smaller\_network = network.copy()

if user\_A != user\_B:

del smaller\_network[user\_A]

for connection in user\_a\_connections:

if connection in smaller\_network:

path = find\_path\_to\_friend(smaller\_network, connection, user\_B)

if path:

return [user\_A] + path

# PASS

# test\_network = create\_data\_structure(example\_input)

# for person in test\_network:

# for clone in test\_network:

# print(person + ' to ' + clone)

# print(find\_path\_to\_friend(test\_network, person, clone))

# Make-Your-Own-Procedure (MYOP)

# -----------------------------------------------------------------------------

# Your MYOP should either perform some manipulation of your network data

# structure (like add\_new\_user) or it should perform some valuable analysis of

# your network (like path\_to\_friend). Don't forget to comment your MYOP. You

# may give this procedure any name you want.

# Replace this with your own procedure! You can also uncomment the lines below

# to see how your code behaves. Have fun!

# net = create\_data\_structure(example\_input)

# print net

# print get\_connections(net, "Debra")

# print get\_connections(net, "Mercedes")

# print get\_games\_liked(net, "John")

# print add\_connection(net, "John", "Freda")

# print add\_new\_user(net, "Debra", [])

# print add\_new\_user(net, "Nick", ["Seven Schemers", "The Movie: The Game"]) # True

# print get\_secondary\_connections(net, "Mercedes")

# print count\_common\_connections(net, "Mercedes", "John")

# print find\_path\_to\_friend(net, "John", "Ollie")

net = create\_data\_structure(example\_input)

print(net)

print(get\_connections(net, "Debra"))

print(get\_connections(net, "Mercedes"))

print(get\_games\_liked(net, "John"))

print(add\_connection(net, "John", "Freda"))

print(add\_new\_user(net, "Debra", []))

print(add\_new\_user(net, "Nick", ["Seven Schemers", "The Movie: The Game"])) # True

print(get\_secondary\_connections(net, "Mercedes"))

print(count\_common\_connections(net, "Mercedes", "John"))

print(find\_path\_to\_friend(net, "John", "Ollie"))

# Error 1, had to change get\_connections and get\_games\_linked to account for person not in network.

network = create\_data\_structure('')

print(get\_connections(network, 'Alice'))

# Error 2, for some reason I was adding users to the network rather than returning False.

network = create\_data\_structure('')

print(add\_connection(network, 'Alice', 'Bob'))

# Expected output: False

# Error 3, was not checking to see if user\_B was already in user\_A's connections.

network = create\_data\_structure('')

network = add\_new\_user(network, 'Alice', [])

network = add\_new\_user(network, "Bob", [])

network = add\_connection(network, 'Alice', 'Bob')

network = add\_connection(network, 'Alice', 'Bob')

print(get\_connections(network, 'Alice'))

# Expected output: ['Bob']