Assignment 4: Finding Connections Between Profiles

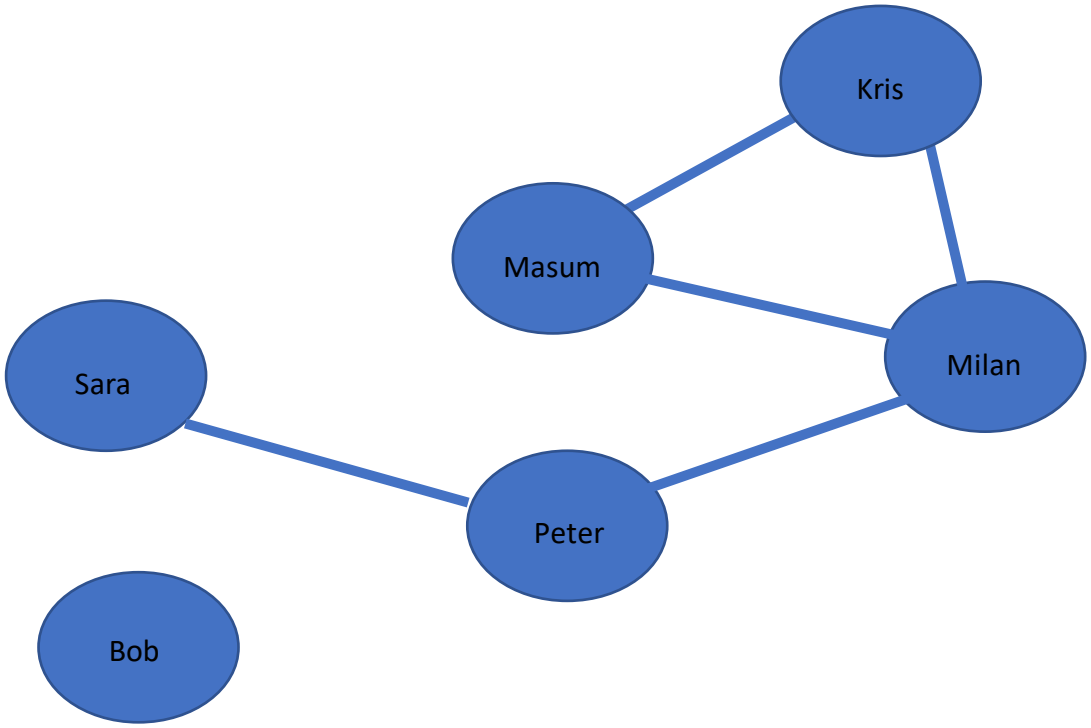
Overview

In this assignment, we practice our Object-Oriented Programming skills by constructing a social network. We'll start by implementing a basic *profile* class to represent a "LinkedIn like" professional network profile. In addition to information about one's current role and employment history, this profile will also maintain a list of connected *profiles*. After completing the implementation of a basic *profile* class, we'll write some functions to work with instances of *profiles.* Required functions include one that answers the question "Where did they work together?" for two profiles and another that answers the question "What is the shortest path between these two profiles?".

More Details

The most (conceptually) difficult part of this assignment is the shortest_path function, so we'll give more details here.

In profile_test.py, notice the connections that are formed……
connect(sara, peter)
connect(peter, milan)
connect(masum, milan)
connect(masum, kris)
connect(milan, kris)
Represented graphically, here are the connections:



We'll describe a basic Breadth First Graph Search Algorithm here in pseudocode and will discuss in class. Keep in mind that we want to return the distance AND the path, so there is some information to keep track of during the search.

```
#def shortest_path(p1, p2):
   #We'll do Breadth first search, keeping track of profiles that we've
   #   already "visited" so that we don't get stuck in cycles.

   #Create a list called visited, initially empty, to keep track of the
   #   profiles that we have already visited. This is important for making
   #   sure that we don't get stuck in a cycle in graph. visited will
   #   simply store a list of profile instances

   #Create a list called q, to keep track of the profiles that we have
   #   not yet explored. Initially, q should be a list that contains a tuple
   #   of 3 things - p1 (the starting node), 0 (representing the distance
   #   from that starting node), and p1.name (the beginning of the path).
   #   More generally, each item in q is a list of tuples, each containing -
   #   a profile, the distance from p1 to that profile, and the path from
   #   p1 to that profile, represented as a list of names. For example,
   #   if p1 is sara, then Milan might eventually get added to q as
   #   (milan, 2, ['sara', 'peter', 'milan']).

   #while q is not empty
   #      grab and remove the 0th item from q
   #      this will give you 3 things. Let's call them...
   #            curr - a profile that we are going to examine
   #            dist - the distance from p1 to curr
   #            path - the path from p1 to curr
   #      add curr to visited
   #      if curr is the thing we're looking for (p2)
   #            we're done! Return the two important things (dist and path)
   #      else
   #             for each of curr's connections
   #                   if the connection is not already in visited
   #                         append a new triple to q
   #                                (the node itself,
   #                                 the distance from p1 to that node - computed from dist,
   #                                 the path from p1 to that node - computed from path)
   #If we hit this point, we never returned an answer. This means that we should
   #      return None because there is no path between p1 and p2.
```

Your job

Implement the profile class as outlined in profile_lib.py. This includes setting up the data members outline in the docstring, as well as implementing the 3 member functions - __init__, __str__ and add_connection.

Implement the 3 functions that are outlined in profile_lib.py:

- connect
- where_did_they_work_together
- shortest_path

Some assert statements are provided in Assignment_4.py but you are welcome to write more asserts.

Complete your work in Assignment_4.py and upload it to canvas.

Extensions

#1 Implement the shortest_path_too_someone_who procedure as outlined in profile_lib.py. This function determines the distance (and associated path) between p1 and someone who meets the criteria expressed in the predicate. Since it is taking a predicate (a function that returns a Boolean) as input, shortest_path_too_someone_who is a “higher order function.”

#2 Merge this work with our chatbot, so that you can ask questions like “Where did Sue and I work together?” or “Who is the closest person in my network that works at Deloitte?”.