ECE241 PROJECT 2: Thinking in Graphs
Due: Nov 5, 2020, 23 PM on Gradescope

Introduction

Online social networks (OSN), such as Facebook, Twitter, and Google+, have gained enormous popularity in the past few years. Millions of users enjoy their lives through online activities like posting their status and pictures, sharing their interests, discussing their thoughts, and communicating with their friends. With the help of those online social networks, you can easily find new friends from your current friends or common interests. In this project, you are tasked to find new potential friends for specific users by analyzing an anonymous social network graph on Facebook.

In the assignment, you will build a graph that represents the relationship between Facebook users. There is an edge between two users if they are friends on Facebook. The weight of the edge is the closeness of two users, which is defined as the number of communications in each common post. Using this graph, you will be able to connect two unacquainted users with those friends' connections. You can also estimate the potential probability of them becoming new friends through their close friends.

Task Overview

In this assignment, you need to accomplish the following specific tasks.

1. Read the Facebook OSN data from file "facebook_network.csv" and build the Facebook user connectivity graph.
    - Create a class named ***OSN***.
    - Create a method ***buildGraph(self, filename: str) -> None:*** to load the data from a filename (with the input parameter as a string). You don't need to return anything but store the graph in a class variable ***self.network***.
    - Please import and use the ***Graph*** and ***Vertex*** classes provided in the class.
    - Each row in the file represents an edge between two Facebook users. The number after the names is the closeness score between them, which is the number of communications in their common posts.
2. Find the shortest distance (minimum number of friends that can connect them) between any two Facebook users.

   - Create a method ***findDistance(self, user1: str, user2: str) -> int*** to find the distance between user1 and user2 and return it (as an integer).
   - You can think about using BFS or a Dijkstra algorithm.
3. Build a best friend sub-graph by generating a maximum spanning tree from the original Facebook user connectivity graph. The best friend relationship of a user is the edge with the highest closeness weight among all the edges to his friends.
   - Create a method ***buildMST(self) -> None***. You don't need to return anything but store the sub-graph in a class variable ***self.MST***.
4. Find the best friend path between two users along the maximum spanning tree.
   - Create a method ***findPath(self, user1:str, user2:str) -> str***
   - For example, for two users David and Alice, the output should be a string like "David -> Bob -> Alice".
5. Find the friend connection path between two users with the highest total closeness score in the original Facebook user connectivity graph. To connect any two users, it's better to have fewer intermediate friends between them. Therefore, this friend connection path should **ONLY consider the path with the shortest distance between two users**.
   - Create a method ***findClosePath(self, user1:str, user2:str) -> str***
   - For two users David and Alice, the output should be "David -> Bob -> Alice (397)". The number in the bracket is the total closeness score among the path.

Hints and suggestions

1. Try to debug your program using a small list of users and closeness scores first. Only use the big, complete file once you feel confident your program works.
2. Try to get portions of the program to work step-by-step. See if you get the shortest path function to work first and then build the MST. You can perform the final two tasks at the very end.
3. The maximum spanning tree is an extension of the minimum spanning tree which is a spanning tree of a weighted graph having maximum weight instead of the minimum weight. You can easily modify the MST code to achieve this goal.
4. Start early. Even though we provide a lot of the basic code from the lectures, it might take a while to write and debug these methods.

What to submit:

Please submit all .py files for your project to Gradescope. All code should be well commented.

Reminder: The course honesty policy requires you to write all code yourself, except for the code that we give to you. Your submitted code will be compared with all other submitted code for the course to identify similarities. Note that our checking program is not confused by changed variable or method names.

Grading:

- Code works (50%)
- Comments (20%)
- Program structure (20%)
- Readability (10%)