CSE 211

Data Structures

TERM PROJECT SPRING 2024

Graph Implementation to Social Network

This is a social network analysis project that implements a graph data structure to represent relationships between people. The graph is composed of vertices, where each vertex represents a person and is represented by a Person object. The edges between the vertices represent relationships between people, and these relationships are stored as a list of friend IDs in each Person object.

The code provides functionality to add people to the graph, get information about a person, suggest friends to a person based on common friends, occupation, or age, calculate degree centrality, clustering coefficient, and use the Girvan-Newman algorithm to find communities in the network.

The code is implemented in C++ and includes the following files:

- **Graph.h**: The header file for the Graph class, which implements the graph data structure and its related functionality.
- **Graph.cpp**: The implementation file for the Graph class.
- **Person.h**: The header file for the Person class, which represents a person in the network.
- **Person.cpp**: The implementation file for the Person class.
- **utils.h**: The header file for the utility functions used by the code. (Not necessary.)
- **utils.cpp**: The implementation file for the utility functions.
- **main.cpp**: The main.cpp file serves as the entry point for the program and contains the implementation of the user interface and interaction with the Graph class to perform various social network analysis tasks.
- **report.pdf**: Write a single page document.

Graph Class: The Graph class is a representation of a graph data structure that stores information about people and their relationships in a social network. The class has functions to add a person to the graph, retrieve a person based on their ID, suggest potential friends to a person, calculate various centrality measures, perform graph partitioning and much more. The class has a private data member graph which is a vector of pairs where the first element of each pair is the person's ID and the second is an instance of the Person class representing the person. The Graph class uses the Person class to store information about a person and their relationships with other people in the graph. The Graph class has the following methods and properties:

- `addPerson(int id, const Person &person)`: a method that takes an integer id and a Person object, and adds it to the graph,
- `getPerson(int id) const`: a method that takes an integer id and returns a pointer to the Person object in the graph with that ID,
- `<< operator`: an overloaded stream operator that allows printing the graph to an output stream,
- `suggestFriends(int person_id, int mode) const`: a method that takes an integer person_id and an integer mode and suggests friends for the person with the given person_id based on the given mode, mode names;
  1.Common Friends,
  2.Similar Occupation,
  3.Similar Age.
- `suggestFriendsByCommonFriends(const Person *person) const`: a method that takes a pointer to a Person object and suggests friends for that person based on the number of common friends they have with other people in the graph,
- `suggestFriendsByOccupation(const Person *person) const`: a method that takes a pointer to a Person object and suggests friends for that person based on the similarity of their occupation,
- `suggestFriendsByAge(const Person *person) const`: a method that takes a pointer to a Person object and suggests friends for that person based on the similarity of their ages,
- `getSize() const`: a method that returns the number of people in the graph,
- `degreeCentrality() const`: a method that calculates the degree centrality of each person in the graph and prints it,
- `clusteringCoefficient(int id) const`: a method that takes an integer id and returns the clustering coefficient of the person with that ID in the graph,
- `getGraph() const`: a method that returns the graph as a vector of pairs, where each pair consists of an integer id and a Person object,
- `girvanNewman(int iterations) const`: a method that takes an integer iterations and returns the communities in the graph using the Girvan-Newman algorithm,
- `edgeWeight(const Graph &graph, int u, int v) const`: a static method that takes a Graph object and two integers u and v and returns the weight of the edge between the two people with IDs u and v in the given graph,
- `removeFriendship(int id1, int id2)`: a method that takes two integers id1 and id2 and removes the friendship between the two people with those IDs in the graph.

The Graph class also has a private property `std::vector<std::pair<int,Person>> graph` that stores the graph as a vector of pairs, where each pair consists of an integer id and a Person object. The class also has a private property `int size` that stores the number of people in the graph.

- There is no Node Structure in this implementation. The `std::pair<int,Person>` is used to represent the mapping between a person's ID and their Person object. The intis used as the key (the person's ID), and the Person object is the value. Therefore, graph is a vector of vectors of pairs, where   each inner vector contains a list of connected nodes represented as pairs of an ID and a Person object.(If you want, you can implement the Node Structure instead of using mapping method.)
- You should use `removeFriendship` and `edgeWeight` functions inside the `girvanNewman` function.

Person Class: The Person class is a data structure that holds information about a person, including their ID, name, age, gender, occupation, and list of friends (also represented as IDs). The class provides methods to access and manipulate this data.

Main: Make a menu-driven command line program that provides various functionalities for analyzing a social network. The social network is represented as a graph data structure, which is read from a CSV file `social_network.csv` using the `readData` function from the `utils.cpp` file.

Menus:
1.Display the social network,
2.Suggest friends for users based on common friends, similar occupation, or similar age,
3.Calculate degree centrality for each user,
4.Calculate clustering coefficient for each user,
5.Detect communities using the Girvan-Newman algorithm,
6.Exit the program.

Girvan Newman Algorithm: The `girvanNewman` function implements the Girvan-Newman algorithm to detect communities ina social network graph. It returns a list of communities, where each community is represented by a list of person IDs (integers).

- Wikipedia Page
- Medium Article

Parameters:

- `int iterations`: The number of iterations the algorithm should run for. Each iteration removes the edge with the highest weight (betweenness) from the graph.

Return value: `std::vector<std::vector<int>>`: A vector of vectors, where each inner vector represents a community and contains the person IDs (integers) of its members.

Function Steps:

1. Create a copy of the current graph.
2. Run the Girvan-Newman algorithm for the specified number of iterations. In each iteration, find the edge with the highest weight (betweenness) and remove it from the graph.
3. Determine the minimum and maximum person IDs in the graph.

4. Create a list of communities, initially empty.
5. For each person in the graph, add their friends to the corresponding community.
6. Remove empty communities from the list.
7. Return the list of communities.

Note: The `edgeWeight` function calculates the betweenness (weight) of the edge between two given nodes in the graph.

Report: A comprehensive one-page documentation of your project should be provided.

- Introduction: Start with a brief introduction to the problem you were trying to solve and the objectives of your code.
- Methodology: Describe the approach you took to solve the problem and the algorithms you used. Explain any new concepts or techniques you learned while working on the project
- Implementation: Detail the steps you took to implement the solution in code. Discuss any trade-offs you made and the reasons behind them. Explain any key decisions you made while writing the code, such as data structures or libraries you used.
- Results: Present the results of your code.
- Conclusion: Summarize the key points of your report and highlight the significance of your findings. Discuss any limitations of your code and suggest areas for future improvement.

Submission

- You should compress(Preferred extension is zip.) all your files (NOT a folder, except **utils.h** & **utils.cpp**) and submit a single file named with your name and student ID, just as you did for your homework, to YULEARN.
    - The zip file name should be in the following format: `<STUDENT_NAME><STUDENT_ID>.zip` for example (batuhanEdgüer2023000000.zip). Just a reminder, your name and surname are not "Batuhan Edgüer".= Do your own work to stay away from punishment.
    - Goodluck!
    - You can ask all your questions via email or you can visit me on Tuesdays.
    - If you make any changes to the functions, you will get zero credit (for the specific question).
    - If your code isn't compiled on my computer, you will get zero credit for your assignment.
    - Commented codes will be ignored.