Group 1 Project

Graph Algorithms for Social Network Analysis

Project Description

Students will apply different algorithmic paradigms to problems in social network analysis:

1. **Greedy Algorithm:** Implement a greedy Influence Maximization Algorithm based on the High-Degree Heuristic.
2. **Divide-and-Conquer:** Implement the Girvan-Newman Algorithm for community detection, which uses a divide-and-conquer approach by recursively removing edges based on their betweenness centrality.
3. **Dynamic Programming:** Implement the Bellman-Ford Algorithm to compute shortest paths in social networks that may have negative edge weights.
4. **Approximation Algorithms:** Implement an approximation algorithm for the Maximum Clique Problem in undirected graphs.

Learning Outcomes

- Understand the application of graph algorithms in analyzing social networks.
- Learn to implement and analyze algorithms for community detection and centrality measures.
- Explore heuristic methods in pathfinding algorithms.
- Tackle NP-hard problems using approximation algorithms in practical contexts.

What It Takes to Execute This Project

Algorithm Design and Analysis

- Understanding of graph theory and social network concepts.

- Knowledge of algorithmic strategies for complex problem-solving.
- Ability to perform algorithmic analysis and optimization.
- Proper selection and justification of data structures used.
- Accurate time and space complexity analysis.
- Comparative discussion of algorithm efficiencies.

Proficiency in Python Programming

- Strong Python programming skills.
- Experience with implementing complex algorithms.
- Familiarity with Python libraries relevant to graph theory (e.g., NetworkX).

Optional Extra Credit (30%)

User Interface Implementation:

- Provide a user interface for the project that allows:
  - Adding and deleting edges.
  - Editing weights.
  - Generating directed or undirected graphs for a given number of nodes and average degree (density of the graph) with random or customized weights.

Delivery Method

- **Written Report:**
  - Theoretical background, methodology, and analysis for each problem.
  - Justification for the selection of data structures.
  - Complexity analysis and discussion of results.
  - Comparative analysis of algorithm efficiencies.
- **Python Code:**
  - Clean, efficient, and well-documented code implementations.
  - Proper use of data structures and programming practices.
  - Readable code with comments explaining key sections.
- **Test Cases:**

  - Real or simulated social network data for testing algorithms.
  - Include comprehensive test cases covering normal and edge cases.
  - Interpret and discuss the results obtained.

- **Presentation:**
  - Discuss findings and their implications in social network analysis.
  - Use visual aids to enhance understanding (e.g., graphs, flowcharts).
  - Present in a clear, organized, and professional manner.

Rubrics for Grading and Evaluation

- **Algorithm Design and Analysis (25%):**
  - Clear and accurate explanations of each algorithm.
  - Proper selection and justification of data structures used.
  - Accurate time and space complexity analysis.
  - Comparative discussion of algorithm efficiencies.
  - Demonstrated understanding of when and why each algorithm is used.

- **Implementation (40%):**
  - Correctness and efficiency of code.
  - Proper use of data structures and programming practices.
  - Code readability and thorough documentation.
  - Effective handling of edge cases and error conditions.

- **Presentation and Report (25%):**
  - Clarity, organization, and professionalism of the written report.
  - Effectiveness in communicating findings and methodologies.
  - Quality of visual aids and examples provided.
  - Insightfulness in discussing findings and their implications in social network analysis.

- **Testing (10%):**
  - Comprehensive test cases covering normal and edge cases.
  - Correct interpretation and discussion of results.
  - Evidence of rigorous testing to ensure algorithm correctness.

- **Optional Extra Credit (Up to 30%):**

- **User Interface Implementation:**
  - Functionality for adding/deleting edges and editing weights.
  - Ability to generate graphs with specified parameters (nodes, density, weights).
  - User-friendly design and interaction.
  - Additional features that enhance usability or visualization.