A GRAPH BASED ON SOCIAL NETWORK

A PROJECT REPORT

Submitted by

MOHANA HARINI.S
(2303811710422097)

in partial fulfillment of requirements for the award of the course

CGA1121 – DATA STRUCTURES

in

COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

June 2024

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report titled "A GRAPH BASED ON SOCIAL NETWORK" is the bonafide work of MOHANA HARINI S (2303811710422097), who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE
Mrs.A.Delphin Carolina Rani,
M.E.,Ph.D.,
HEAD OF THE DEPARTMENT
PROFESSOR Department of CSE
K. Ramakrishnan College of Technology
(Autonomous)
Samayapuram–621112.

SIGNATURE
D.P.Devan B.E., M.E.,
ASSISTANT PROFESSOR
Department of CSE
K. Ramakrishnan College of
Technology (Autonomous)
Samayapuram–621112.

INTERNAL EXAMINER

Submitted for the viva-voce examination held on 15/06/2024

DECLARATION

I declare that the project report on "A GRAPH BASED ON SOCIAL NETWORK" is the result of original work done by us and best of our knowledge, similar work has not been submitted to "ANNA UNIVERSITY CHENNAI" for the requirement of Degree of BACHELOR OF ENGINEERING. This project report is submitted on the partial fulfillment of the requirement of the award of the course CGA1121- DATA STRUCTURES.

Signature

MOHANA HARINI S

Place: Samayapuram

Date: 15/06/2024

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and indebtedness our institution, K. Ramakrishnan College of Technology (Autonomous)", for providing us with the opportunity to do this project.

I extend our sincere acknowledgment and appreciation to the esteemed and honorable Chairman, Dr. K. RAMAKRISHNAN, B.E., for having provided the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director, Dr. S. KUPPUSAMY, MBA, Ph.D., for forwarding our project and offering an adequate duration to complete it.

I would like to thank Dr. N. VASUDEVAN, M.TECH., Ph.D., Principal, who gave the opportunity to frame the project to full satisfaction.

I thank Dr. A. DELPHIN CAROLINA RANI M.E., Ph.D., Head of the Department of COMPUTER SCIENCE AND ENGINEERING, for providing her encouragement in pursuing this project.

I wish to convey our profound and heartfelt gratitude to our guide D.P.DEVANB.E., M.E., Assistant Professor / CSE Department of COMPUTER SCIENCE AND ENGINEERING incalculable suggestions, creativity, assistance and patience, which motivated us to carry out this project.

I render our sincere thanks to the Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education.

MISSION OF THE INSTITUTION

- Produce smart technocrats with empirical knowledge who can surmount the global challenges.
- Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students.
- Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations.

VISION OF DEPARTMENT

To be a center of eminence in creating competent software professionals with research and innovative skills.

MISSION OF DEPARTMENT

M1: Industry Specific: To nurture students in working with various hardware and software platforms inclined with the best practices of industry.

M2: Research: To prepare students for research-oriented activities.

M3: Society: To empower students with the required skills to solve complex technological problems of society.

PROGRAM EDUCATIONAL OBJECTIVES

1. PEO1: Domain Knowledge

To produce graduates who have strong foundation of knowledge and skills in the field of Computer Science and Engineering.

2. PEO2: Employability Skills and Research

To produce graduates who are employable in industries/public sector/research organizations or work as an entrepreneur.

3. PEO3: Ethics and Values

To produce the ethics and values of the world challenges.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Domain Knowledge

To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO 2: Quality Software

To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO 3: Innovation Ideas

To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1.Engineering knowledge:
Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2.Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

3.Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations

4.Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5.Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

Social networks represent intricate webs of relationships among individuals, necessitating robust computational frameworks for analysis and visualization. This abstract outlines a project focusing on leveraging graph theory and data structures to explore and derive insights from social network data.The project utilizes graph data structures where users are nodes and relationships (friendships, interactions) are edges, facilitating efficient representation and traversal. Key algorithms such as BFS and DFS enable exploration of network connectivity, while centrality measures like Degree Centrality and Betweenness Centrality identify influential users and bridges within the network. Community detection algorithms further reveal cohesive groups or clusters based on interaction patterns.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGENO.
	ABSTRACT	vii
	LIST OF FIGURES	i x
	LIST OF ABBREVIATIONS	xi
1	INTRODUCTION	
	1.1 Introduction	1
	1.2 Purpose And Importance	1
	1.3 Objectives	2
	1.4 Project Summarization	2
2	PROJECT METHODOLOGY	
	2.1. Introduction to System Architecture	3
	2.2 Detailed System Architecture Diagram	4
3	DATA STRUCTURE PREFERANCE	
	3.1 Explanation of why a Doubly Linked List was chosen	5
	3.2 Comparison with Other Data Structures	6
	3.3 Advantages and disadvantages of using a DLL	6
4	DATA STRUCTURE METHODOLOGY	
	4.1. Adjacency List	7
	4.2. Node Structure	7
	4.3. Initialization, Insertion & Deletion	8
5	MODULES	
	5.1 Create a new contact	9
	5.2 Display the saved contact	9
	5.3 Search the contact	9
6	CONCLUSION & FUTURE SCOPE	
	6.1 Conclusion	10
	6.2 Future Scope	10
	APPENDICES	
	Appendix A-Source code	
	Appendix B -Screen shots	

LIST OF FIGURES

FIGURE NO	TITLE	PAGENO.
2.1	Architecture Diagram	4

LIST OF ABBREVIATIONS

ABBREVIATIONS

DLL	-	Doubly Linked List User
UI	-	Interface Command Line
CLI	-	Interface Graphical User
GUI	-	Interface Natural Language
NLP	-	Processing Graphical User
GUI	-	Interface Application
APIs	-	Programming Interface

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO PROJECT

This project aims to develop a graph-based social network platform.Graph databases are particularly well-suited for social networks because they naturally represent relationships and connections between users. This project will explore the benefits of graph databases, such as improved performance for querying complex relationships, and provide a scalable solution for managing large-scale social networks.

1.2 PURPOSE AND IMPORTANCE OF THE PROJECT

The purpose of integrating data structures in a graph-based social network is to efficiently model and manage the intricate network of connections and relationships among users. By leveraging data structures like adjacency lists or matrices, the system aims to streamline operations such as friend recommendations, content sharing, and personalized user experiences. These data structures facilitate the organized storage, retrieval, and manipulation of user connections, enabling smooth interactions within the network.

The importance of utilizing data structures in a graph-based social network lies in optimizing performance, scalability, and user engagement. These structures help in effectively representing complex relationships between users, allowing for quick retrieval and processing of relevant information. By employing efficient data structures, the system can enhance response times for operations like friend suggestions or content recommendations, leading to improved user satisfaction and retention.

1.3 OBJECTIVES

1.Efficient Data Storage
2.Faster Information Retrieval
3.Enhanced Performance
4.Scalability
5.Data Integrity
6.Improved User Experience

1.4 PROJECT SUMMARIZATION

The project aims to analyze and extract insights from a social network using graph theory and data structures.The project leverages graph theory and data structures to provide in-depth analysis and insights into social network dynamics, offering valuable applications in recommendation systems, anomaly detection, and community management.This summary encapsulates the core aspects of a graph-based social network project, emphasizing its analytical capabilities and practical applications in understanding and leveraging social connections.

a) Graph Representation: Users are represented as nodes, and relationships (friendships, interactions) as edges in a graph structure. This representation facilitates efficient traversal and analysis of the social network.

b) Graph Traversal: Implements BFS and DFS for exploring connections between users, identifying paths, and discovering communities.

c) Graph Visualization: Visualizes the network structure to intuitively understand user relationships and community formations.

d) Scalability: Evaluates the scalability of algorithms and data structures to handle large-scale social network datasets.

e) Accuracy:Validates algorithmic outputs against ground truth data or benchmarks, ensuring reliable insights and analysis.

CHAPTER 2

PROJECT METHODOLOGY

2.1 INTRODUCTION TO SYSTEM ARCHITECTURE

In the realm of modern social networking platforms, the architecture supporting the underlying graph-based structures plays a pivotal role in enabling efficient data management, real-time analytics, and seamless user experiences. This introduction explores the fundamental aspects of system architecture tailored for graph-based social networks.

2.1.1 High-Level System Architecture

The high-level architecture of the social network graph system comprises several major components:

(i)User Interface (UI)

(ii)Application Server

(iIi)Graph Data Store

2.1.2 Components of System Architecture

Social networks inherently involve complex relationships and connections between users, best represented and managed through graph databases.Effective system architecture for graph-based social networks is indispensable for delivering a responsive, scalable, and feature-rich user experience. By leveraging specialized graph databases, microservices, advanced analytics, and real-time processing capabilities, these architectures empower social platforms to not only manage but also derive actionable insights from the complex interconnections that define modern social interactions.

a) Tools: Utilizes Apache Kafka for real-time data streaming and Apache Spark for batch processing, ensuring timely updates and data consistency.

b) Design: Decomposes monolithic applications into loosely coupled microservices, each responsible for specific functionalities (e.g., user management, recommendation engines).

c) Functionality: Executes graph algorithms (e.g., centrality measures, community detection) to derive insights about user behavior, influence, and community structures.

2.2 DETAILED SYSTEM ARCHITECTURE DIAGRAM

Include a diagram that visually represents the system architecture. The diagram should depict how each component interacts with the others. For example, it can show the User Interface sending requests to the Application Logic, which in turn interacts with the Data Management Layer and the Storage Layer.

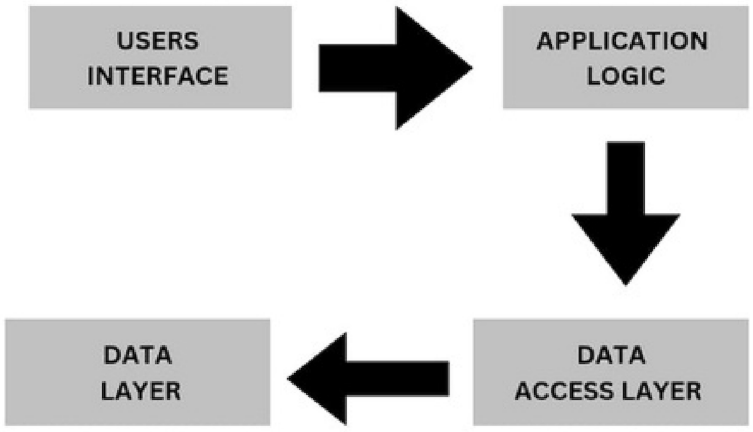

Fig 2.1 : Architecture Diagram (Sample)

CHAPTER 3

DATA STRUCTURE PREFERANCE

3.1 EXPLANATION OF WHY ADJACENCY LIST, ADJACENCY MATRIX WAS CHOSEN

The adjacency list was chosen for representing the graph-based social network due to its efficient space utilization, dynamic modification capabilities, and suitability for traversal-heavy operations. Its alignment with the sparse nature of social networks and the practical advantages it offers in real-world applications underscore its appropriateness for this project. Choosing an adjacency matrix for representing a graph-based social network is driven by its ability to provide constant-time edge lookups, facilitate efficient implementation of graph algorithms, and support dense network structures effectively. While it may incur higher space complexity in sparse networks, its benefits in operational efficiency, simplicity, and suitability for dense graphs make it an appropriate choice for certain social network analysis projects.

a) Memory Usage: In an adjacency list, each node has a list of adjacent nodes (friends or connections). This uses O(V + E) space, where V is the number of vertices (users) and E is the number of edges (connections), making it highly memory-efficient for large, sparse networks.

b) Flexibility: They accommodate dynamic changes without the need to resize or restructure the entire representation, unlike adjacency matrices.

c) Fixed Size: The fixed size of the matrix (V x V) provides predictable memory usage and simplifies implementation, as the structure does not change dynamically.

d) Uniformity: The uniform structure of adjacency matrices can simplify certain operations and ensure consistent access patterns, reducing the complexity of code related to edge management.

e) High Connectivity: If the social network is dense, with a large number of connections relative to the number of users, the adjacency matrix becomes more space-efficient. In such cases, the O(V^2) space complexity is justified as most entries represent actual edges.

3.2 COMPARISON WITH OTHER DATA STRUCTURES

In the context of social networks, representing the relationships between users effectively is crucial for both storage efficiency and computational performance. The two primary data structures for graph representation are adjacency lists and adjacency matrices, each with its own advantages and trade-offs. Below is a detailed comparison of these data structures along with edge lists, another common representation.

3.3 ADVANTAGES AND DISADVANTAGES OF USING AL

3.3.1 Advantages of Using Adjacency List:

Using adjacency lists for representing a graph-based social network provides significant advantages in terms of space efficiency, dynamic edge management, traversal efficiency, and alignment with real-world network structures. Their scalability, simplicity, and flexibility make them an ideal choice for handling the complex and evolving nature of social networks, ensuring efficient performance and ease of development for a wide range of graph-based applications. Adjacency lists can handle real-time updates efficiently, making them ideal for applications that require instantaneous reflection of user activities, such as social media feeds, live interaction tracking, and real-time recommendation systems.

3.3.2 Disadvantages of Using a Adjacency List:

While adjacency lists offer significant advantages for representing sparse graphs, their inefficiencies in edge lookup, handling dense graphs, implementing certain algorithms, and managing real-time updates present notable disadvantages. In dense social networks or applications requiring frequent edge checks and complex global graph operations, adjacency lists may not be the optimal choice. These limitations highlight the importance of selecting the appropriate data structure based on the specific characteristics and requirements of the social network being analyzed.

CHAPTER -4

DATA STRUCTURE METHODOLOGY

4.1 ADJACENCY MATRIX

An adjacency matrix is a fundamental data structure used to represent a graph-based social network. In this matrix, each row and column correspond to users, and the cell at the intersection indicates whether a connection exists between them. For an undirected graph, the matrix is symmetric. This structure allows for quick edge existence checks with constant time complexity, making it efficient for dense graphs where many connections are present. Despite its simplicity and ease of implementation, the adjacency matrix's O(n^2) space complexity can be a drawback for large, sparse social networks, where many matrix entries remain unused. However, its straightforward representation and rapid access to edge information make it a valuable tool for analyzing and managing social networks with dense connections.

4.1.1 FEATURES OF ADJACENCY MATRIX

1. Matrix Representation:An adjacency matrix is a 2D array of size (n \times n), where (n) is the number of nodes (users) in the graph.The element at row (i) and column (j) (i.e., (\text{matrix}[i][j])) indicates the presence (and possibly weight) of an edge between node (i) and node (j).
2. Time Complexity:Edge existence checks and edge additions or deletions are performed in (O(1)) time, making these operations very efficient.
3. Space Complexity:The adjacency matrix requires (O(n^2)) space, which can be substantial for large networks. This makes it more suited to dense graphs where the number of edges is close to the number of possible edges (n^2).
4. Implementation Simplicity:The structure is straightforward to implement and understand, as it relies on basic array manipulation.
5. Dense Network Efficiency:The adjacency matrix is particularly efficient for dense social networks where most users are interconnected, as it allows for efficient storage and retrieval of edge data.
6. Memory Usage:While the adjacency matrix is not memory efficient for sparse graphs, its memory usage is predictable and constant, making it useful in contexts where memory size can be managed and is not a constraint.

4.2 NODE STRUCTURE

When designing a graph-based social network in a data structure, the node (representing a user) is a crucial component.This node structure provides a comprehensive and flexible framework for representing users in a social network graph. It allows for efficient management of user connections and interactions, supporting various graph operations and analyses. By encapsulating user data and relationships, this structure facilitates the development of robust social network applications.

Breadth-First Search (BFS): Explore nodes level by level; find the shortest path in unweighted graphs.Use a queue to manage the exploration frontier.

Depth-First Search (DFS): Explore as far as possible along each branch before backtracking; useful for pathfinding and cycle detection.Use a stack (or recursion) to manage the exploration frontier.

CHAPTER-5
MODULES

5.1 User Interface (UI):

- Handles user interactions and displays data.
- Key components: Login/Register, Profile, News Feed, Friend List, Messaging, Notifications, Search.

5.2 Application Logic:

- User Management: Handles user authentication and profile updates.
- Post Management: Manages creation, editing, and deletion of posts.
- Friend Management: Manages friend requests and connections.
- Message Management: Manages sending and receiving messages.
- Notification: Manages user notifications.
- Search: Provides search functionality and recommendations.

5.3 Data Access Layer:

- Graph Database Interface: Manages interactions with the graph database.
- Cache Management: Optimizes data retrieval with caching mechanisms.

5.4 Data Layer:

- Graph Structure: Represents users and relationships using nodes and edges.
- User Data: Stores user information.
- Post Data: Stores post content and metadata.
- Message Data: Stores messages between users.
- Notification Data: Stores notifications for user activities.

CHAPTER 6
CONCLUSION & FUTURE SCOPE

6.1 CONCLUSION

In conclusion, representing social networks as graphs within data structures provides profound insights into the intricate web of user interactions and relationships. By leveraging adjacency lists and matrices, one can efficiently model and analyze the network's connectivity, centrality, and community structure. These analyses not only illuminate the key influencers and densely connected communities but also reveal dynamic changes over time. The of such graph-based analyses are vast, ranging from recommendation systems to targeted marketing, while also addressing challenges like scalability and user privacy. Ultimately, the intersection of graph theory and data structures equips us with powerful tools to decode the complexities of social networks, driving innovations in technology and social media.

6.2 FUTURE SCOPE

The future scope for graph-based analysis of social networks using data structures is vast and promising. With the exponential growth of social media data, advancements in data structures and algorithms will be crucial for efficiently processing and analyzing large-scale networks. Enhanced graph databases and distributed computing techniques will facilitate real-time analysis and visualization of complex networks. Machine learning and AI integration will further refine predictive models for user behavior, community detection, and anomaly detection. Additionally, addressing privacy concerns through advanced cryptographic methods and privacy-preserving algorithms will become increasingly important. As the interplay between social networks and data structures evolves, we can expect breakthroughs in personalized recommendations, social influence modeling, and the overall understanding of human social dynamics, driving innovation across various domains such as marketing, healthcare, and cybersecurity.

APPENDICES

APPENDIX A-SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAME_LENGTH 100

// Structure to represent a node (user) in the graph
typedef struct User {
char name[MAX_NAME_LENGTH];
struct User* next;
} User;

// Structure to represent an adjacency list node
typedef struct AdjListNode {
struct User* user;
struct AdjListNode* next;
} AdjListNode;

// Structure to represent an adjacency list
typedef struct AdjList {
AdjListNode* head;
} AdjList;

// Structure to represent a graph
typedef struct Graph {
int numUsers;
User* users;
AdjList* array;
} Graph;

// Function to create a new user node
User* createUser(char* name) {
User* newUser = (User*)malloc(sizeof(User));
strncpy(newUser->name, name, MAX_NAME_LENGTH);
newUser->next = NULL;
return newUser;
}

// Function to create a new adjacency list node
AdjListNode* createAdjListNode(User* user) {
AdjListNode* newNode = (AdjListNode*)malloc(sizeof(AdjListNode));
newNode->user = user;
newNode->next = NULL;
return newNode;
}
```

```
// Function to create a graph
Graph* createGraph() {
Graph* graph = (Graph*)malloc(sizeof(Graph));
graph->numUsers = 0;
graph->users = NULL;
graph->array = NULL;
return graph;
}

// Function to add a user to the graph
void addUser(Graph* graph, char* name) {
// Create a new user node
User* newUser = createUser(name);

// Add the new user to the user list
newUser->next = graph->users;
graph->users = newUser;
graph->numUsers++;

// Reallocate memory for the adjacency list array
graph->array = (AdjList*)realloc(graph->array, graph->numUsers * sizeof(AdjList));
graph->array[graph->numUsers - 1].head = NULL;
}

// Function to get a user by name
User* getUser(Graph* graph, char* name) {
User* temp = graph->users;
while (temp) {
if (strncmp(temp->name, name, MAX_NAME_LENGTH) == 0) {
return temp;
}
temp = temp->next;
}
return NULL;
}

// Function to add an edge (friendship) between two users
void addEdge(Graph* graph, char* srcName, char* destName) {
User* srcUser = getUser(graph, srcName);
User* destUser = getUser(graph, destName);
if (srcUser == NULL || destUser == NULL) {
printf("One or both users not found.\n");
return;
}

int srcIndex = -1, destIndex = -1;
User* temp = graph->users;
for (int i = 0; i < graph->numUsers; ++i) {
if (temp == srcUser) {
srcIndex = i;
}
```

```
        if (temp == destUser) {
            destIndex = i;
        }
        temp = temp->next;
    }

    // Add an edge from src to dest
    AdjListNode* newNode = createAdjListNode(destUser);
    newNode->next = graph->array[srcIndex].head;
    graph->array[srcIndex].head = newNode;

    // Since the graph is undirected, add an edge from dest to src as well
    newNode = createAdjListNode(srcUser);
    newNode->next = graph->array[destIndex].head;
    graph->array[destIndex].head = newNode;
}

// Function to print the adjacency list representation of the graph
void printGraph(Graph* graph) {
    User* tempUser = graph->users;
    int i = 0;
    while (tempUser) {
        printf("User %s: ", tempUser->name);
        AdjListNode* tempAdj = graph->array[i].head;
        while (tempAdj) {
            printf("%s ", tempAdj->user->name);
            tempAdj = tempAdj->next;
        }
        printf("\n");
        tempUser = tempUser->next;
        i++;
    }
}

// Function to free the memory allocated for the graph
void freeGraph(Graph* graph) {
    for (int i = 0; i < graph->numUsers; ++i) {
        AdjListNode* tempAdj = graph->array[i].head;
        while (tempAdj) {
            AdjListNode* next = tempAdj->next;
            free(tempAdj);
            tempAdj = next;
        }
    }

    free(graph->array);

    User* tempUser = graph->users;
    while (tempUser) {
        User* next = tempUser->next;
        free(tempUser);
```

```
// Function to free the memory allocated for the graph
void freeGraph(Graph* graph) {
for (int i = 0; i < graph->numUsers; ++i) {
AdjListNode* tempAdj = graph->array[i].head;
while (tempAdj) {
AdjListNode* next = tempAdj->next;
free(tempAdj);
tempAdj = next;
}
}

free(graph->array);

User* tempUser = graph->users;
while (tempUser) {
User* next = tempUser->next;
free(tempUser);
tempUser = next;
}

free(graph);
}

int main() {
Graph* graph = createGraph();

addUser(graph, "Alice");
addUser(graph, "Bob");
addUser(graph, "Charlie");
addUser(graph, "David");

addEdge(graph, "Alice", "Bob");
addEdge(graph, "Alice", "Charlie");
addEdge(graph, "Bob", "David");
addEdge(graph, "Charlie", "David");

printGraph(graph);

// Free allocated memory
freeGraph(graph);

return 0;
}
```

OUTPUT


User David: Charlie Bob
User Charlie: David Alice
User Bob: David Alice
User Alice: Charlie Bob
=== YOUR PROGRAM HAS ENDED ===