Graph-Based Social Network Analysis (C Project)

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

#define MAX_USERS 100

typedef struct Node {
    int user;
    struct Node* next;
} Node;

typedef struct Graph {
    int numUsers;
    Node* adjList[MAX_USERS];
} Graph;

typedef struct Queue {
    int items[MAX_USERS];
    int front, rear;
} Queue;

Graph* createGraph(int users);
void addFriendship(Graph* graph, int user1, int user2);
void displayGraph(Graph* graph);
void findShortestPath(Graph* graph, int start, int end);
void suggestFriends(Graph* graph, int user);
Queue* createQueue();
void enqueue(Queue* q, int value);
int dequeue(Queue* q);
int isQueueEmpty(Queue* q);

int main() {
    int users = 6;
    Graph* graph = createGraph(users);

    addFriendship(graph, 0, 1);
    addFriendship(graph, 0, 2);
    addFriendship(graph, 1, 3);
    addFriendship(graph, 1, 4);
    addFriendship(graph, 2, 4);
    addFriendship(graph, 3, 5);
    addFriendship(graph, 4, 5);

    printf("Social Network Graph:\n");
    displayGraph(graph);

    printf("\nShortest Path from User 0 to User 5:\n");
    findShortestPath(graph, 0, 5);

    printf("\nFriend Recommendations for User 0:\n");
    suggestFriends(graph, 0);

    return 0;
}

Graph* createGraph(int users) {
    Graph* graph = (Graph*)malloc(sizeof(Graph));
    graph->numUsers = users;
    for (int i = 0; i < users; i++) {
        graph->adjList[i] = NULL;
    }
    return graph;
```

```
}

void addFriendship(Graph* graph, int user1, int user2) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->user = user2;
    newNode->next = graph->adjList[user1];
    graph->adjList[user1] = newNode;

    newNode = (Node*)malloc(sizeof(Node));
    newNode->user = user1;
    newNode->next = graph->adjList[user2];
    graph->adjList[user2] = newNode;
}

void displayGraph(Graph* graph) {
    for (int i = 0; i < graph->numUsers; i++) {
        Node* temp = graph->adjList[i];
        printf("User %d -> ", i);
        while (temp) {
            printf("%d ", temp->user);
            temp = temp->next;
        }
        printf("\n");
    }
}

void findShortestPath(Graph* graph, int start, int end) {
    int visited[MAX_USERS] = {0};
    int prev[MAX_USERS];
    for (int i = 0; i < MAX_USERS; i++) prev[i] = -1;

    Queue* queue = createQueue();
    enqueue(queue, start);
    visited[start] = 1;

    while (!isQueueEmpty(queue)) {
        int current = dequeue(queue);
        Node* temp = graph->adjList[current];

        while (temp) {
            int neighbor = temp->user;
            if (!visited[neighbor]) {
                visited[neighbor] = 1;
                prev[neighbor] = current;
                enqueue(queue, neighbor);

                if (neighbor == end) {
                    break;
                }
            }
            temp = temp->next;
        }
    }

    int path[MAX_USERS], pathSize = 0, crawl = end;
    while (crawl != -1) {
        path[pathSize++] = crawl;
        crawl = prev[crawl];
    }

    if (pathSize == 1) {
        printf("No path found.\n");
    } else {
        for (int i = pathSize - 1; i >= 0; i--) {
            printf("%d ", path[i]);
        }
        printf("\n");
```

```
    }
}

void suggestFriends(Graph* graph, int user) {
    int mutual[MAX_USERS] = {0};
    Node* temp = graph->adjList[user];

    while (temp) {
        Node* friendTemp = graph->adjList[temp->user];
        while (friendTemp) {
            if (friendTemp->user != user) {
                mutual[friendTemp->user]++;
            }
            friendTemp = friendTemp->next;
        }
        temp = temp->next;
    }

    int suggested = 0;
    for (int i = 0; i < graph->numUsers; i++) {
        if (mutual[i] > 0 && !mutual[user]) {
            printf("User %d (Mutual Friends: %d)\n", i, mutual[i]);
            suggested = 1;
        }
    }

    if (!suggested) {
        printf("No new friend recommendations.\n");
    }
}

Queue* createQueue() {
    Queue* q = (Queue*)malloc(sizeof(Queue));
    q->front = q->rear = -1;
    return q;
}

void enqueue(Queue* q, int value) {
    if (q->rear == MAX_USERS - 1) return;
    if (q->front == -1) q->front = 0;
    q->items[++q->rear] = value;
}

int dequeue(Queue* q) {
    if (q->front == -1 || q->front > q->rear) return -1;
    return q->items[q->front++];
}

int isQueueEmpty(Queue* q) {
    return q->front == -1 || q->front > q->rear;
}
```