Social Network System Documentation

Overview
This is a C++ implementation of a social network system that allows users to create accounts, manage friendships, search for other users, and handle friend requests. The system is built using custom data structures and provides core social networking functionalities.

System Architecture

Core Components

1. **Graph Template Class** (`Graph.h`)
   - A template-based graph implementation for storing and managing network connections
   - Uses adjacency lists to represent connections between nodes
   - Supports basic graph operations like adding nodes and edges
   - Implemented using custom data structures (Dequeue, LinkedList)

2. **Social Network Class** (`socialNetwork.h`)
   - Main class that handles all social network operations
   - Uses Graph<User*> to store user network
   - Implements friend suggestions using BFS algorithm
   - Manages friend requests and user interactions

3. **User Class** (`User.h`)
   - Represents individual users in the network
   - Stores user information (username, password, age)
   - Manages pending and received friend requests using HashTable
   - Provides methods for user data access and modification

Key Features

1. User Management
- User registration with validation
- Login system with credential verification
- Profile viewing and modification
- Age restriction (minimum 18 years)

2. Friend System
```cpp
void addEdge(int Node1, int Node2)
void removeEdge(int Node1, int Node2)
void handleRequests(const int currentUser, const int userIndex, const int action)
```

- Send/receive friend requests
- Accept/decline friend requests
- Remove friends
- View friend lists

3. Search and Discovery
```cpp
void search(string search, int currentUser)
void peopleYouMayKnow(int currentUser)
```
- Search users by username
- "People You May Know" suggestions using BFS algorithm
- Mutual friends calculation
- Degree-based friend suggestions

4. Social Graph Navigation
```cpp
void BFS(int currentUser, CustomMap<int, int>& Visited)
```
- Breadth-First Search implementation for network traversal
- Finds connections up to 4 degrees of separation
- Calculates mutual friends and connection strength

Data Structures

Custom Implementations
1. **Dequeue**
   - Double-ended queue implementation
   - Used for storing and managing lists of users/friends

2. **HashTable**
   - Custom hash table implementation
   - Used for storing pending and received friend requests

3. **LinkedList**
   - Custom linked list implementation
   - Used in graph adjacency lists

4. **CustomMap**
   - Key-value pair storage
   - Used for tracking visited nodes in BFS

Usage Example

```cpp
int main() {
  SocialNetwork mySocialNetwork;

  // Create users
  User n1("username", "password", 21);
  mySocialNetwork.users.addNode(&n1);

  // Add friendships
  mySocialNetwork.users.addEdge(0, 1);

  // Main application loop
  while (true) {
    // Login/Signup choice
    // User interactions
    // Feature selection (search, view friends, etc.)
  }
}
```

System Limitations and Considerations

1. **Memory Management**
  - Uses pointer-based data structures
  - Requires careful memory handling
  - Potential for memory leaks if not properly managed

2. **Security**
  - Basic password storage (no encryption)
  - No input sanitization
  - Meant for educational purposes

3. **Scalability**
  - In-memory data storage
  - Limited by available system memory
  - O(n) search operations

Implementation Details

Friend Suggestion Algorithm
The system uses a modified BFS algorithm to suggest friends:
1. Traverses the social graph up to 4 degrees
2. Assigns weights based on connection degree
3. Prioritizes mutual friends

4. Excludes existing friends and pending requests

Request Handling
```cpp
void handleRequests(currentUser, userIndex, action) {
   users.get(currentUser)->getRequests().remove(userIndex);
   users.get(userIndex)->getPending().remove(currentUser);
   if (action == 1) {
      users.addEdge(currentUser, userIndex);
   }
}
```

Code Organization

```
project/
├── Graph.h        # Graph template class
├── socialNetwork.h  # Main social network implementation
├── User.h         # User class definition
├── Social.cpp     # Main application entry point
└── Graph.cpp      # Graph implementation details
```

Future Improvements

1. **Security Enhancements**
  - Password hashing
  - Input validation
  - Session management

2. **Performance Optimization**
  - Improved search algorithms
  - Caching mechanisms
  - Better memory management

3. **Feature Additions**
  - User posts/content sharing
  - Privacy settings
  - Group functionality

4. **Data Persistence**
  - Database integration

- File-based storage
- Backup/restore functionality

Conclusion
This social network implementation provides a solid foundation for understanding graph-based social networks and custom data structure implementation in C++. While it has limitations, it successfully demonstrates core social networking concepts and algorithms.