Comsians Social Media App

M. Ahmad Raza -> SP21-BCS-003

Deepak Kumar -> SP21-BCS-017

Malik Rashid Ali -> SP21-BCS-041

Date: 3rd July 2022

Contents

Introduction:

We are trying to solve the huge communication gap between two or more people at the same time. Global connection from strangers and structured directions from one person to another. To solve all these problems, we make this project. This app can connect friends irrespective of their cultures, ethnicity, and any value. Every user can also see suggestions from friends of friends to make their friend circle vast and strong. It is also able to show structured shortest path decided from one city to another.

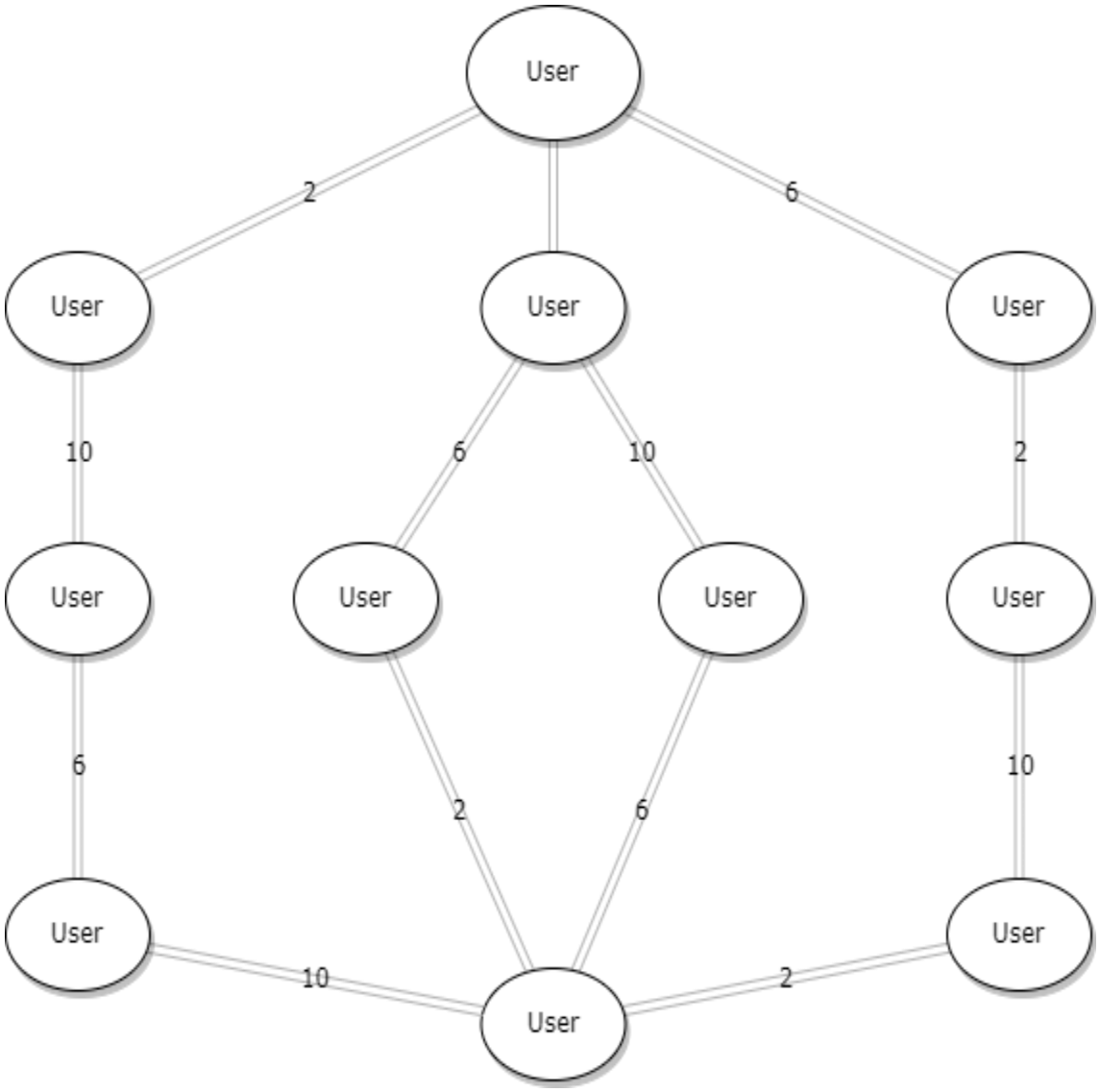
Features:

- End on end and user to user Interaction.
- Users can post anything, can see all posts of themselves and others.
- Users can also like the posts of other users.
- Users can also get suggestion of adding friends of friends

Data Structures Used:

- Array
- Linked list
- 2D Array
- Queue
- Graph
- HashMap
- Set

Visual Representation:



What have we done?

Signup:

```
void signUp(User *user)
{
    Node *node = new Node;
    node->user = user;
    for (int i = 0; i < SIZE; i++)
    {
        if (matrix[i][0] == NULL)
        {
            matrix[i][0] = node;
            return;
        }
    }
}
```

Explanation:

If a user wants to join our social media app, he must create an account. The above method is signup method of our code which takes user of User* type as an argument. This method traversing the adjacency matrix and search for NULL index whenever it finds a NULL index it places the given user at the respective index.

Send Request:

```
void send_request(User *sender, User *receiver)
{
    int sender_index = search(sender);
    int receiver_index = search(receiver);

    if (sender_index == -1 || receiver_index == -1)
    {
        cout << "User not found\n";
        return;
    }

    matrix[receiver_index][0]->user->requests.push(sender);
}
```

Explanation:

After login or signup user can send friend request to any other user of this app. For this, this method takes sender* User and receiver* User as arguments then search for both User* if they exit just pull request by pushing sender* User in the queue<*User> present in the struct of the user.

Incoming Request Process:

```
void requests(User *user)
{
    if (user->requests.empty())
    {
        cout << "\nNO REQUESTS FOUND\n";
        return;
    }
    cout << "NAME: " << user->requests.front()->name << endl;
    cout << "USER NAME IS: " << user->requests.front()->username << endl;

    cout << "\nTO ACCEPT ---> Press y \n TO REJECT ---> Press n\n";
    cout << "ENTER YOUR CHOICE: ";
    string option;
    cin >> option;

    if (option == "y")
    {
        addFriend(user, user->requests.front());
        user->requests.pop();
    }
    else if (option == "n")
    {
        user->requests.pop();
    }
    else
    {
        cout << "\nWRONG CHOICE\n";
        requests(user);
    }
}
```

Explanation:

After sending a request, other users also check their request list by using this method. If there is no request found just show a message of “no request found” otherwise, it’ll pop requests from his\her own queue list from struct.

Posts:

```
void addPost(User *user)
{
    string post;
    cout << "Add a posts: ";
    cin.ignore();
    getline(cin, post);
    Post *newPost = new Post;
    newPost->myPost = post;
    user->posts.push_back(newPost);
    cout << "\nPOST ADDED\n";
}
void showMyPost(User *user)
{
```

```
    display(user->posts);
}
bool postIdVerification(string id)
{
    for (int i = 0; i < id.size(); i++)
    {
        if (!isdigit(id[i]))
        {
            return false;
        }
    }
    return true;
}
```

Explanation:

User can also add posts and show his/her post to all the user who are friends of this user on this app.

Friend Suggestions Processing:

```
void friendSuggestions(User *user)
{
    unordered_set<string> names;
    int userIndex = search(user);
    if (matrix[userIndex][1] == NULL)
    {
        cout << "NO FRIEND SUGGESTIONS LIST IS AVAILABLE\n";
        return;
    }
    for (int i = 1; i < SIZE; i++)
    {
        if (matrix[userIndex][i] == NULL)
        {
            break;
        }
        else
        {
            int friendIndex = search(matrix[userIndex][i]->user);
            if (friendIndex == -1)
            {
                cout << "NO FRIEND SUGGESTIONS LIST IS AVAILABLE\n";
                return;
            }
            for (int j = 1; j < SIZE; j++)
            {
                if (matrix[friendIndex][j] == NULL)
                {
                    break;
                }
                else if (!myFriendExists(matrix[friendIndex][j]->user-
>username, userIndex))
                {
```

```
pushHashMap(matrix[friendIndex][j]->user->username);
                    names.insert(matrix[friendIndex][j]->user->username);
                }
            }
        }
    }
    Que q(100);
    unordered_set<string>::iterator it;
    for (it = names.begin(); it != names.end(); ++it)
    {
        hashMap *map = new hashMap;
        map->frequency = hash_map[getAsciValues(*it)];
        map->value = *it;
        q.enqueue(map);
    }
    q.display();
    cout << "ENTER USER NAME TO SEND REQUEST(0 to return): ";
    string recevierName;
    cin >> recevierName;
    if (recevierName == "0")
    {
        return;
    }
    User *recevier = search(recevierName);
    if (recevier != NULL)
    {
        send_request(user, recevier);
        friendSuggestions(user);
    }
    else
    {
        cout << "RECIEVER NOT FOUND";
        friendSuggestions(user);
    }
}
```

Explanation:

This app also can show the suggestions for friends of friends by using the above piece of code. This whole procedure reflects the actual idea of graphs. If two users are connected and the third user relates to one of them then this method show the other user to the third one

Like And Dislike of Post:

```
void likePost(User *user, string usernameInput, string id)
{
    if (!postIdVerification(id))
    {
        cout << "<<<<<<<<<<<<<<< INVALID POST ID >>>>>>>>>>>>>>>" << endl;
        return;
    }
    int postId = stoi(id);
    int liked = search(search(usernameInput));
```

```
    if (liked == -1)
    {
        cout << "USER NOT FOUND";
        return;
    }
    Post *post = NULL;

    int count = 1;

    if (matrix[liked][0]->user->posts.size() < postId)
    {
        cout << "\n <<<<<<<<<<<<<<< INVALID POST ID >>>>>>>>>>>>>> \n";
        return;
    }
    for (auto const &item : matrix[liked][0]->user->posts)
    {
        post = item;
        if (count >= postId)
        {
            break;
        }
        count++;
    }
    post->likes++;
    post->postUser.push_back(user);

    int liker = search(user);
    for (int i = 1; i < SIZE; i++)
    {
        if (matrix[liked][i] != NULL)
        {
            if (matrix[liked][i]->user->username == user->username &&
                matrix[liked][i]->weight > 2)
            {
                matrix[liked][i]->weight -= 4;
            }
        }
        if (matrix[liker][i] != NULL)
        {
            if (matrix[liker][i]->user->username == matrix[liked][0]->user-
>username &&
                matrix[liker][i]->weight > 2)
            {
                matrix[liker][i]->weight -= 4;
            }
        }
    }
}

void dislike(User *user, string usernameInput, string id)
{
    if (!postIdVerification(id))
    {
        cout << "<<<<<<<<<<<<<<< INVALID POST ID >>>>>>>>>>>>>>>" << endl;
        return;
    }
    int postId = stoi(id);
```

```
    int liked = search(search(usernameInput));
    if (liked == -1)
    {
        cout << "USER NOT FOUND";
        return;
    }
    Post *post = NULL;

    int count = 1;

    if (matrix[liked][0]->user->posts.size() < postId)
    {
        cout << "\n <<<<<<<<<<<<<<<< INVALID POST ID >>>>>>>>>>>>>>> \n";
        return;
    }
    for (auto const &item : matrix[liked][0]->user->posts)
    {
        post = item;
        if (count >= postId)
        {
            break;
        }
        count++;
    }
    post->dislikes++;
    post->postUser.push_back(user);

    int liker = search(user);
    for (int i = 1; i < SIZE; i++)
    {
        if (matrix[liked][i] != NULL)
        {
            if (matrix[liked][i]->user->username == user->username)
            {
                matrix[liked][i]->weight += 4;
            }
        }
        if (matrix[liker][i] != NULL)
        {
            if (matrix[liker][i]->user->username == matrix[liked][0]->user-
>username)
            {
                matrix[liker][i]->weight += 4;
            }
        }
    }
}
```

Explanation:

This app also can allow user to like and dislike the posts of other users by this we can have the intentions of user's friendship closeness with each other. These both are the interesting feature we have introduced by using the matrix that was basically having all info of that user, so that he can decide what he want to do in his feed.