Object Oriented Programming

MTE Project Report

Aryan(2K19/SE/019) and Bhavishya(2K19/SE/028)

Social Network Database

Submitted to: Mr. Rohit Beniwal

Abstract

The application aims at providing a simple social(people) network database system to keep track of inter-personal relationships(friends or incompatible) of people and give suggestion for distribution of people in teams/groups accordingly.

The application uses graph data structure at its core with various features that may be useful to an organization(like dual mode). Dual mode, one for the ‘Admin’ and the other for the ‘user’ helps satisfy the requirement of both(described further in the text).

The application uses graph methodology to save and store the link information though our implementation is a bit different as we made use of file a lot. The two modes are described below :

1. User mode : The user mode is mainly for data entry and for user to check the status(group, contribution score etc) of their friends in the company. When in user mode, the user is first required to enter their user id(given at the time of making the account) and password, he/she can then do the following things :

*View and modify their data/status

*View their friends status

*Delete their account

2. Admin Mode : The admin mode is designed for the head of the team/company as so as to ensure that each and every team has sufficient human resources and least number of members have conflicting opinions(discordants). There is only one Admin account thus when the admin option is selected only password in required to enter. The admin then has the following options :

*Who is the most compatible member(by number of friends)

*Statistics for each member(skills, compatibility)

*determining mutual friends or chain of friends for surveys

*Ability to add/delete any account (but still cannot modify it)

The application is targetted for large group management and team management in a company and is meant to be ran on a server where some users may or may not be able to access.

Table of contents

Objective

Graph Implementation

Traditionally the graphs are of two types :

1. Adjacency Matrix : here the linkage information is stored in 2-dimensional array(or matrix) of boolean, each (x,y) element of the stores 0 or 1 depending whether there is a link between x node and y node or not. Limitation is that it cannot have a dynamic size/state thus could not be used for our program.

2. Adjacency list : here the is a list/array of pointers thus can have dynamic state and by using datatypes like vector we can even implement dynamic size. But as we were unaware of any way of storing this data to a file as each time different memory is allocated.

Thus we developed our own way using three files :

one for storing the information of various participants to the disc, second for storing the linkage information, third for storing some temporary data for troubleshooting and testing.

There are two classes(mainly) one for the data of user and other for linkage(not implemented yet thus, may be changed)

Working of all this is explained further in the documentation.

Implement file I/O for permanent data storage

As mentioned above we user mainly two files :

1. “usrs.dat”(binary file as the data cannot be easily accessed by anyone) for storing the the personal and statistical data of the users, this file stores the elements of the class user (may refer some modules attached with the document) and helps to restore the state by also reloading the state variables like the count of users recent modifications/deletions etc.

2. “link.txt” the file implementation is still under development and may be changed, this file is used to store the linkage information of the users. The information stored in the file can be processed to work just like a graph but with some limitations which may lead to more files and higher depenedence on the disk speed rather than the memory(which may be good for low memory

devices) thus we are also trying to add a way to also use the memory of the computer to balance the load. Some snippets of code contain this as well.

Writing algorithms for power tools for the admin

We are yet to write the algorithms and methods for the admin thus all the data presented will be just approaches : here we plan to implement this at the end when the graphs and storage algorithms(hard part) is functioning optimally as then it seems not too tough to implement the algorithms like BFS, DFS etc. As the algorithms are already known to me. My teammate is yet to study these algorithms thus we will be distributing that work as soon as we can.

Description

As mentioned above, the project has two modes admin and user mode. In user mode, user's data include name, password, skill sets as there are three skills of every user and user id. Skills help in determining their ability to work in a group as the users having same skill sets can do work with each other or you can say they can be placed in the same group. Similarly skill sets also help in determining discordants as no discordant should be in any group.

In this project, there are 4 modules that are

1. Administrative
2. Adding a user
3. Login
4. Display user

Administrative: this function is houses the admin menu accessible by entering a password that is hardcoded into the program code for security reasons. Anyone who knows the administrative password can make use of the powerfull tools provided by the program such as finding links and suggest groups that takes in account all the social relations of a person and the skill set they have to judge whether they are compatible enough to fit in a perticular group requiring a certain skill set.

Adminstrative password is admin

```
bool admin_algos()
{
        int ans, y[3], t, i;                    // for user input to menu
//      infor.seekg(-sizeof(temp1), ios::end);
//      if (!infor.good()) {cout << "not good";}
//      infor.read((char *) & temp1, sizeof(temp1));
//      user_no = temp1.uid;
//      cout << usr[0].uid;
   do
   {
//     system("clear");
                cout << "\n\t\tSOCIAL NETWORK DATABASE\n\n";
                cout << "1.display all links\n2.most liked and disliked user\n3.suggest groups\
n4.\n\n==>"; cin >>ans;
                switch(ans)
                {
                        case 1:
                                display_links();
                                break;
                        case 2:
                                most_links();
                                break;
                        case 3:
                                cout << "\nenter the required skill-set below:\n";
```

```
                                cout << "0->production\t1->research\t2->marketing\t3->human_resource\t4->accounting & finance\nenter the following in accordance :";
                                for (i = 0; i<3 && y[i-1] != -1; i++) {
                                        cout << "\nenter 0 or 1 or 2 or 3 or 4 or -1(to stop).....";
                                        cin>>y[i];
                                }
                                if(i == 1)
                                        mkgroup(y[0]);
                                else if (i == 2)
                                        mkgroup(y[0], y[1]);
                                else if (i == 3)
                                        mkgroup(y[0], y[1], y[2]);
                                break;
                        case 4:
                                break;
                }
        }while(ans == 1 || ans == 2 || ans == 3 || ans == 4);
        return 1;
}
```

Display_links : this function refers to the all the objects and in addition to displaying their information, it also displays the information of their friends and discordants. The program works on the following algorithm.

It first goes to a particular object of array of class user after displaying the data of that user it goes to the next friend whose address is stored in the data member friend_link pointer and prints the corresponding linked list(as the program makes use of adjacency list).

The same process(loop) is ran for printing the discordants except this time instead of the friend_link, disc_link pointer is used which stores the linked list containing the uids of all discordants.

```
void display_links()                    // to display all the links when in admin mode
{
        fgraph_node *ftemp;
        cgraph_node *ctemp;
        for (int i = 0; i <= user_no; i++) {
                usr[i].disp_info();
                ftemp = usr[i].friend_link;
                cout << "\n\n\tthe user has the following friends :";
                while (ftemp != nullptr) {
                        cout<< "\n\t"; usr[ftemp->uid].disp_info();
                        ftemp = ftemp->friend_link;
                }
                ctemp = usr[i].disc_link;
                cout << "\n\tthe user has the following discordinators :";
                while (ctemp != nullptr) {
                        cout<< "\n\t"; usr[ctemp->uid].disp_info();
                        ctemp = ctemp->disc_link;
                }
                cout << "\n\n";
        }
}
```

Most links : this functions works in a very similar way as the above display_links function except it does not display the data instead it increments count variables to display the count of the friends and discordants a participant has.

The program works by going at each object of array of class user and the going to each user's friend and discordant linked list and incrementing the fcount and ccount variables respectively,

then the comparison is made between fmax and fcount and the bigger value is stored in fmax, similarly the comparison is made between cmax and ccount and the bigger value is stored in cmax, both are stored with their corresponding indices.

At the end the user with most friends and discordants is displayed respectively.

We would have further modified the program to display most and least compatible user, but due to the time constraints we weren't able to.

```
void most_links()
{
        fgraph_node *ftemp;
        cgraph_node *ctemp;
        int fmax = 0, cmax = 0, temp1, temp2;
        for (int i = 0; i <= user_no; i++) {
                int fcount = 0, ccount = 0;
                ftemp = usr[i].friend_link;
                while (ftemp != nullptr) {
                        fcount++;
                        ftemp = ftemp->friend_link;
                }
                ctemp = usr[i].disc_link;
                while (ctemp != nullptr) {
                        ccount++;
                        ctemp = ctemp->disc_link;
                }
                if (fcount > fmax) {fmax = fcount; temp1 = i;}
```

```
            if (ccount > cmax) {cmax = ccount; temp2 = i;}
        }
        cout <<"the following user has the most friends :\n";
        usr[temp1].disp_info();
        cout << "       with " << fmax << " friends\n";
        cout <<"the following user has the most discordants :\n";
        usr[temp2].disp_info();
        cout << "       with " << cmax << " discordants\n";
}
```

mkgroup : this function is the main focus of our program as it tells us about the users who work in a group together having similar skill set and most compatibility with each other(as compatibility is subjective thus it needs to be calculated everytime the group creation is required).

The function works by initially taking the input for skill set(requirement) and runs the loop based on the number of the parameters provided:

depending on the parameter provided the function only considers that and chooses the user on the basis of that.

All the choosen users are stored in the gruid array which contains their uids and compatibility. If any of the chosen user are friends or discordants with any other chosen user then the compatibility is incremented and decremented respectively

if the chosen users(j) are greater than the number of users required(strength) then the user with the least compatibility is placed at the end of the group(j) and then j is decremented, the whole process again repeats till the number of users chosen is equal to the strength of the group.

```
Void mkgroup(int req1, int req2 = -1, int req3 = -1)        // req is for requirement
{
        int i = 0, strength, j = -1, k;
        gmem gruid[30], temp;
        cout << "\nhow many people do you want in the group => ";
        cin>>strength;
        if (req2 == -1 && req3 == -1) {
                for (i = 0; i <= user_no; i++) {
                        if (usr[i].check_skill(req1)) {j++; gruid[j].uid = i;}
                }
        }
        else if (req3 == -1) {
                for (i = 0; i <= user_no; i++) {
                        if (usr[i].check_skill(req1) && usr[i].check_skill(req2)) {j++;
gruid[j].uid = i;}
                }
        }
        else {
```

```
                for (i = 0; i <= user_no; i++) {
                        if (usr[i].check_skill(req1) && usr[i].check_skill(req2) &&
usr[i].check_skill(req3)) {j++; gruid[j].uid = i;}
                }
        }
        if (j == -1) {cout << "no user has the required skill-set\n"; return;}
        for (k = 0; k <= j; k++) {
                for (i = 0; i <= j; i++) {
                        if (is_friend(gruid[k].uid, gruid[i].uid)) {gruid[i].compatiblity++;}
                        if (is_discordant(gruid[k].uid, gruid[i].uid)) {gruid[i].compatiblity--;}
                }
        }
        while (j > strength-1) {
                gmem min;
                int index_of_min = 0;
                min = gruid[0];
                for (int i = 0; i <= j; i++) {
                        if (gruid[i].compatiblity < min.compatiblity) {index_of_min = i; min =
gruid[i];}
                }
                gruid[index_of_min] = gruid[j];
                gruid[j] = min;
                j--;
        }
        cout << "\nRecommended group members are as follows :\n";
        for (int i = 0; i <= j; i++) {
//                cout << gruid[i].uid <<"\t" << i << "\n";              // just for debugging
                cout << "\t";
                usr[gruid[i].uid].disp_info(); cout << endl;
        }
}
```

Adding a user: In this module a new user is added or a new account has formed. So user first have to give his name, and set password that should not be more than 50 words. Then there is option for choosing his/her skills from given skill sets. After that user id will be allotted to the user and user account added successfully. User id will be used at time of login.

```
bool add_user()
{
	fgraph.open("friend_graph.txt", ios::in | ios::ate | ios::out);
	cgraph.open("discordant_graph.txt", ios::in | ios::ate | ios::out);
	infor.open("usrs.dat", ios::binary | ios::in | ios::ate | ios::out);
	user u;
	user_no++;                                  // incrementing as a new user is added
	int input_uid;
	bool temp;
	usr[user_no].input(user_no);
	cout << "\nenter the uids of users the user is friends with(type -1 to exit) : \n" <<
"friend's uid => ";
	cin>>input_uid;
	while (input_uid != -1) {
	// here fgraph is the friend graph's fstream object
		if (!save_link(usr[user_no].uid, input_uid, fgraph)) {cout << "Link saving
failed"; break; return 0;}
		if (!f_create_link(usr[user_no].uid, input_uid)) {cout << "link creation in
memory failed"; break; return 0;}
		cout << "friend's uid => ";
		cin>>input_uid;
	}
	cout << "\nenter the uids of users the user is discordant with(type -1 to exit) : \n" <<
"discordant's uid => ";
	cin>>input_uid;
	while (input_uid != -1) {
	// here cgraph is the discordant graph's fstream object
```

```
                if (!save_link(usr[user_no].uid, input_uid, cgraph)) {cout << "Link saving failed"; break; return 0;}
                if (!c_create_link(usr[user_no].uid, input_uid)) {cout << "link creation in memory failed"; break; return 0;}
                cout << "discordant's uid => ";
                cin>>input_uid;
        }
        fgraph.close(); cgraph.close();
        infor.seekp(0, ios::beg);
        if (infor.good()) {
                infor.seekp(0, ios::end);
                u = usr[user_no];
                infor.write((char *) &u, sizeof(u));
                infor.close();
        }
        else {cout << "File access failed\n"; infor.close(); return 0;}
        return 1;
}
```

Login: In this module user has to login with their user id and password. Then there will be option to modify their account as users can update their data that are asked while creating account and can also add friend or delete friend to from their user account. Similarly they can also add discordant or remove discordant from their discordants list.

Login constitutes two parts modifying profile and deleting user id.

```
case 3:
                cout << "enter your user id(UID) => "; cin>>t;
                cout<<”enter password => ”;
                cin>>password;
                if(!check_pass(password)) {
                        cout<<”password incorrect”;
                        break;
                }
                usr[t].disp_info();
                int ch;
                cout<<"\n1. edit my user profile "<<endl;
                cout<<"2. delete my id "<<endl;
                cout<<"Enter your choice(1 or 2)"<<endl;
                cin>>ch;
                switch(ch)
                {
                   case 1: modify_user(t);
                           break;
            case 2: delete_user();
                  break;
                  }
                  break;
```

modify_user: This function helps in modifying user's data and also helps in adding friend or removing friend and adding or removing discordant. Firstly it give user three choices as 1. To add or remove a friend 2. To add or remove a discordant 3. Edit password and skill set 4. End.

when user opt for 1st choice then they have two choices more that are 1. To add friend 2. To remove friend, they can choose any option for their requirements. Similarly if user opt for choice 2 then they can add or delete discordant. Third choice is to edit user's password and skill set. 4. Choice is to end function.

```
void modify_user(int userid)
{
   int choice, choice_f, choice_d,friend_id,discordant_id;
   cout<<"\n1. Add or Remove a friend "<<endl;
   cout<<"2. Add or Remove a discordant "<<endl;
   cout<<"3. edit my skills and password "<<endl;
   cout<<"enter your choice -> ";
   cin>>choice;
   if (choice==1)
   {
      cout<<"enter user id of friend: ";
      cin>>friend_id;
      cout<<"1. add friend"<<endl;
      cout<<"2. remove friend"<<endl;
      cout<<"enter your choice"<<endl;
      cin>>choice_f;
      if(choice_f==1)
      {
         if(save_link(userid,friend_id, fgraph))
            cout<<"friend added successfully"<<endl;

      }
      else
```

```
    {
        if(delete_link(userid,friend_id, fgraph))
            cout<<"friend removed successfully"<<endl;
    }
}
if(choice == 2)
{
    cout<<"enter user id of discordant: "<<endl;
    cin>>discordant_id;
    cout<<"1. add discordant"<<endl;
    cout<<"2. remove discordant:"<<endl;
    cin>>choice_d;
    if(choice_d==1)
    {
        if(save_link(userid, discordant_id, cgraph))
            cout<<"discordant added successfully"<<endl;
    }
    else
    {
        if(delete_link(userid, discordant_id, cgraph))
            cout<<"discordant removed successfully"<<endl;
    }
}
else
{
    int user_id;
    user usr;
    fstream file3;
    file3.open("usrs.dat", ios::in | ios::out | ios::binary);
    cout<<"enter your user id"<<endl;
    cin>>user_id;
    while( file3.read((char *)&usr, sizeof(usr)))
```

```
        {
            if( usr.return_userid()== user_id)
            {
                usr.input( user_id);
                file3.seekg( -sizeof(usr), ios::cur);
                file3.write( (char *)&usr, sizeof(usr));
            }
        }
    }
}
```

Delete_user : This function helps in deleting user id of a user. It first asks for user id that a user want to delete and then it will be deleted. Firstly function asks user to enter user id that user want to delete and then from all the records in binary file except the record with entered user id, all copied to another file and then the file is going to rename the same name as it has previously.

```
bool delete_user()
 {
    int user_id;
    user usr;
    int input_uid;
    bool temp1;

     //int user_id;
    cout<<"Enter user id of user you want to delete"<<endl;
    cin>>user_id;

  //  cout<<" Enter the user id the user is friend with(type -1 to exit)"<<endl;
   // cin>>input_uid;

    /*    temp1= delete_link(usr.uid, user_id,fgraph);
        if(temp1 ==1)
        {
            cout<<"link is deleted"<<endl;
            return 0;
        }
        else
           cout<<"user id is not matched, hence link is not deleted"<<endl;

   // cout<<" Enter the user id the user is concordant with(type -1 to exit)"<<endl;
   // cin>>input_uid;

        temp1= delete_link(usr.uid, user_id,cgraph);
        if(temp1 ==1)
```

```
        {
            cout<<"link is deleted"<<endl;
            return 0;
        }
        else
          cout<<"user id is not matched, hence link is not deleted"<<endl;*/


    fstream file1;
    fstream file2;
    file1.open("usrs.dat", ios::in |ios::binary);
    file2.open("usrs_temp.dat", ios::out | ios::binary);



    while ( file1.read((char *)&usr, sizeof(usr)))
     {
        if(usr.return_userid()!= user_id)
          file2.write((char *)&usr, sizeof(usr));
     }
     file1.close();
     file2.close();
     remove("usrs.dat");
     rename("usrs_temp.dat","usrs.dat");
}
```

save_link : this function has only one job that is to copy newly formed link onto the disk for permanent storage,

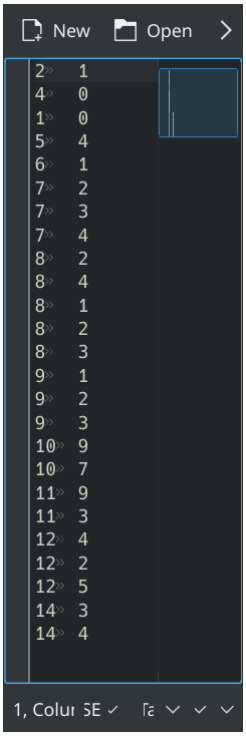
this uses a file in which the storing process was totally thought by us the image shows how the links are stored in memory.

```
bool save_link(int to_uid, int input_uid, fstream &file)
{
        file.seekp(0, ios::end);
        if (file.good()) {file << to_uid << "\t" << input_uid << endl;}
        else {return 0;}
        return 1;
}

bool delete_link(int to_uid, int input_uid, fstream &file)
{
   int u_id;
   file.seekp(0, ios::beg);
   cout<<"enter user id of user who want to remove"<<endl;
   cin>>u_id;
   while(file.eof())
   {
      if(u_id == to_uid)
      {
         file.seekp(-6, ios::cur);
         file<<"   "<< "\t" <<"   "<<endl;
         return 1;
      }

   }

}
```



Display users : In this module, all the users will be displayed with their details as user id , name, skill sets etc. This module is for admin mode to see the user and also to group them together.

```
void display()
{
//	user usr;
//	infor.seekg(0, ios::beg);
//	while (infor.good())
//	{
//		infor.read((char *) &usr, sizeof(usr));
//		usr.disp_info();
//	}
	for (int i = 0; i <= user_no; i++) {
		usr[i].disp_info();
		cout << endl;
	}
}
```

Classes in our project: There are three classes in our project that are:

1. fgraph_node
2. cgraph_node
3. user

In fgraph _node there is a an int data member that is uid and a pointer variable which is pointing to its own class type variable i.e. it stores the address of another variable which store their friend's user id.

In cgraph_node , similar to fgraph node there is int data member uid and also a pointer variable which stores the address of variable with discordant user id.

User class contains all the data that are needed at the time of creating account namely name, password, skill sets etc. it also contains member functions that are input to take input from user and allocating user id, disp_info to display all details of users, check_pass to check the password correct or not, operator and return_userid which returns the user id of user for whom it is invoked.

Files: There are three files, two text files and a binary file.

1. fgaph- text file
2. cgraph- text file
3. infor- binary file

fgaph file contains the links between friends or you can say user's friend regarding information.

cgaph file contains the link between discordants or you can say user's discordant regarding information.

infor file contains user details as it is binary file save data in record form so all the record of user(details) in the company will be saved in infor binary file.

Functions: and their purpose

All the functions are listed below.

display: In this function, data of user is displayed by disp_info member function of class user as it is invoked in this finction.

save_link: In this function, what will going to happen is it saved the link between two user in cgraph or fgraph file as when a user add friend or add discordant.

add_user: This function helps in adding user while creating an account, this works for user mode. This function first ask the details of user and allot user id. Then it also make the link between user and their friend after it asks for friends user id. Similarly also asks for discordant and make the link between them.

f_create_link: this function is used for creating a link in the friend graph within the objects in the memory(i.e. user objects)

c_create_link: this function is used for creating a link in the discordant graph within the objects in the memory(i.e. user objects)

initialize_database:this function initializes all the data stored on non-volatile storage and used that to initialize the database i.e. copy all the data on the disk to objects of user class in memory and making the links between them.

display_links: This function will display all the links that are present between any two user either they are friend or discordant. First of all it displays user information and then it displays all their friend and all their discordants.

delete_link: This function delete the link between two users which is required when any user want to delete friend or discordant. So this function helps in deleting friend or discordant.

delete_user: This function helps in deleting user id of a user. It first asks for user id that a user want to delete and then it will be deleted.

modify_user: This function helps in modifying user’s data and also helps in adding friend or removing friend and adding or removing discordant.

is_friend: to check whether a user has the other user as a friend

is_discordant: to check whether a user has the other user as a discordant

Execution

The code and the binary files(combined in a zip file) are attached with the project report and thus, the program can be executed by running the binary file(.exe) in the extracted folder(formed after extracting the zip file).

Note : in the program all the sample accounts provided have their name as their password and the adminstrative password is admin

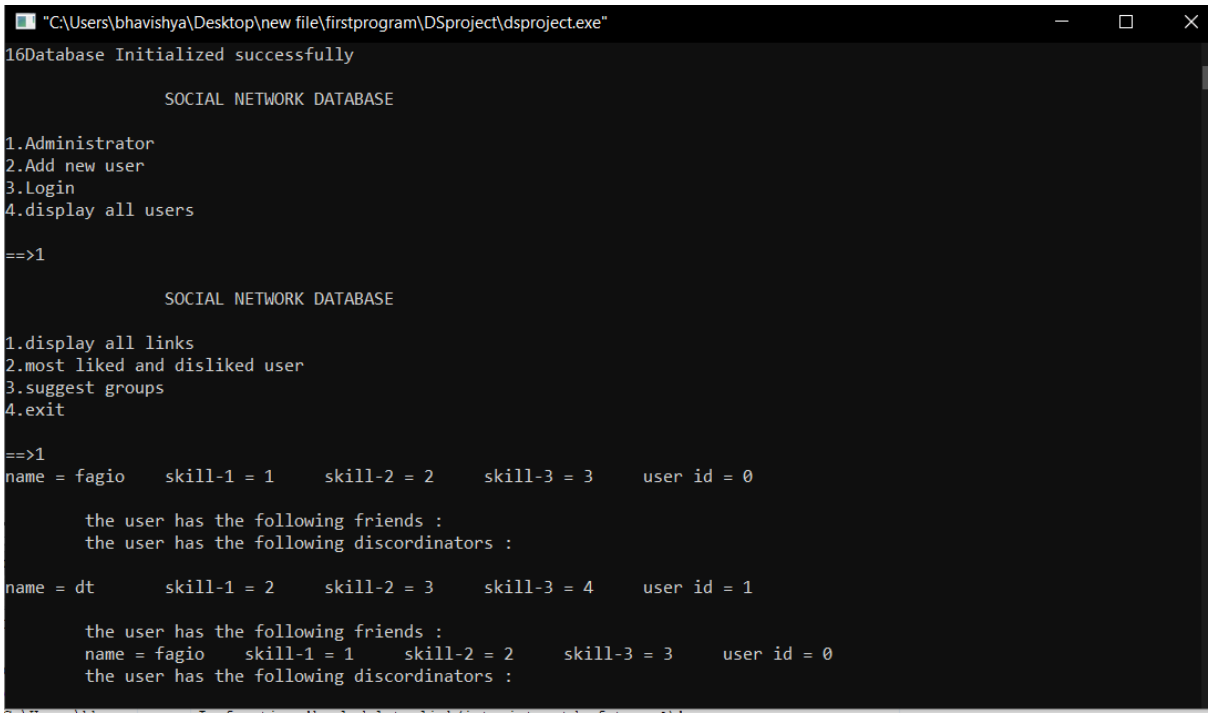
In this section, we will going to describe the working of project i.e. what is going to happen when the program is going to run.

First of all user has the following choices

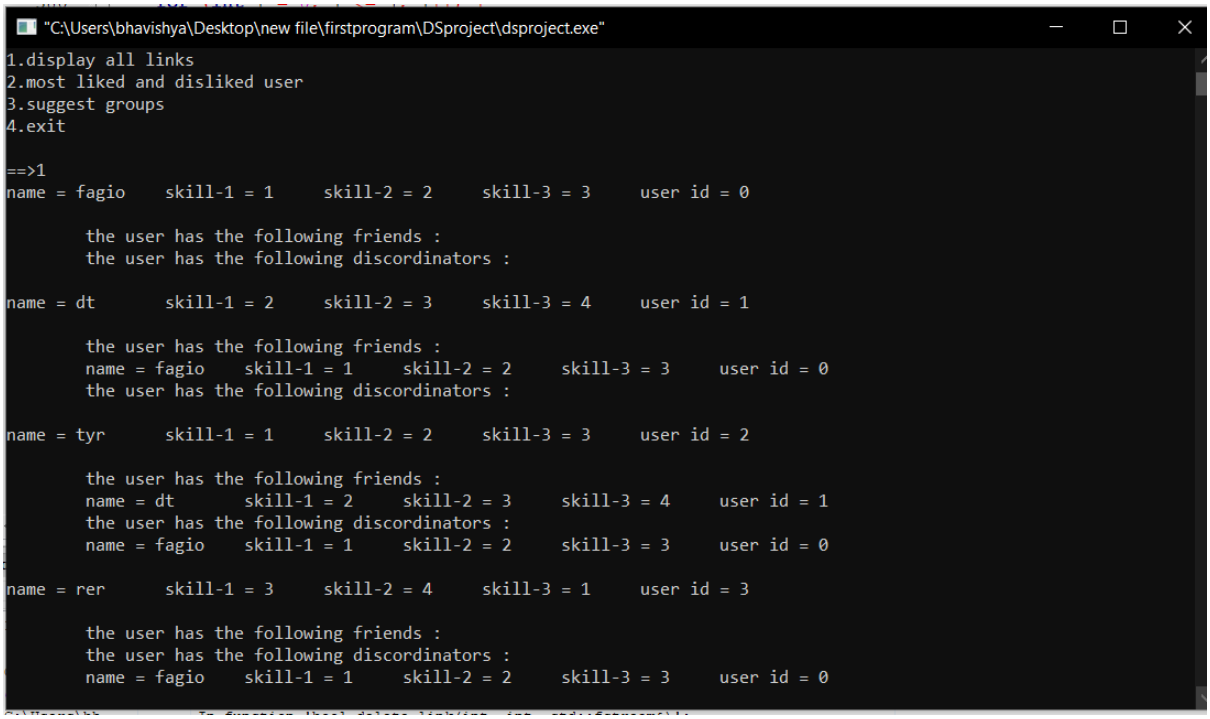
1. Administrator
2. Add new user
3. Login
4. Display all user

After opting for choice **1. Administrator** the new interface has the following choices.

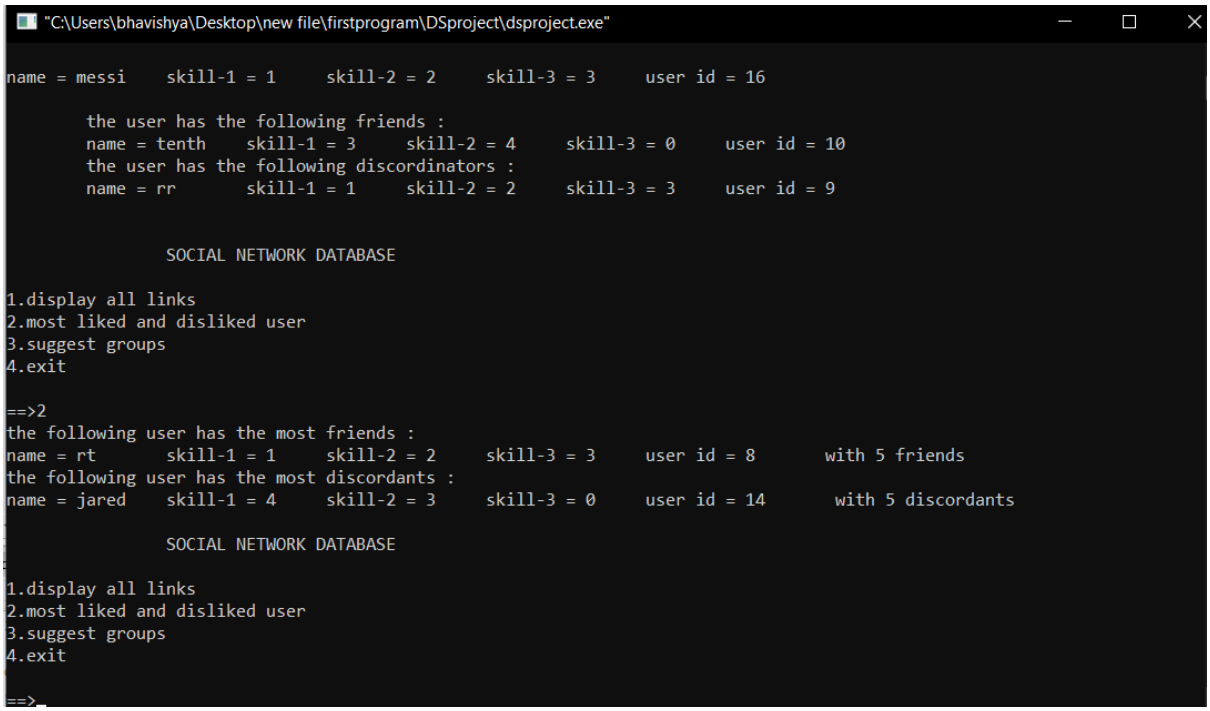
1. Display all links
2. Most liked and disliked user
3. Suggested groups
4. End



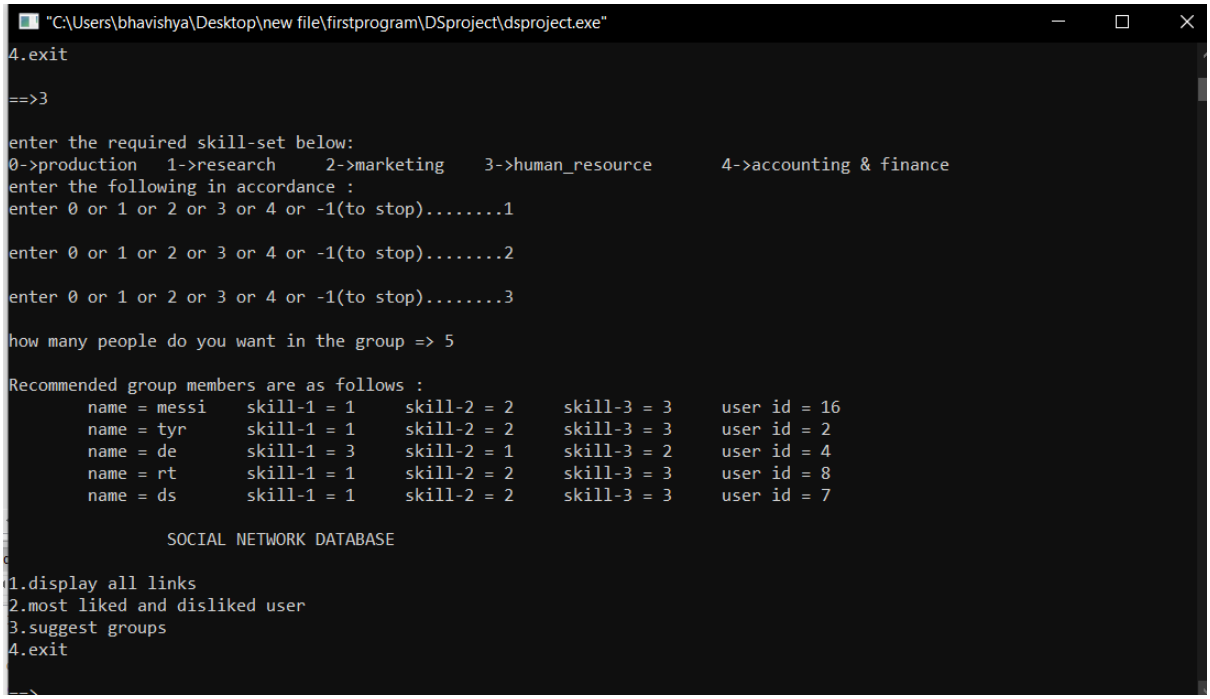
If user opt for choice 1. Display all links then all the links of all the users present in the database is visible on screen at that time.



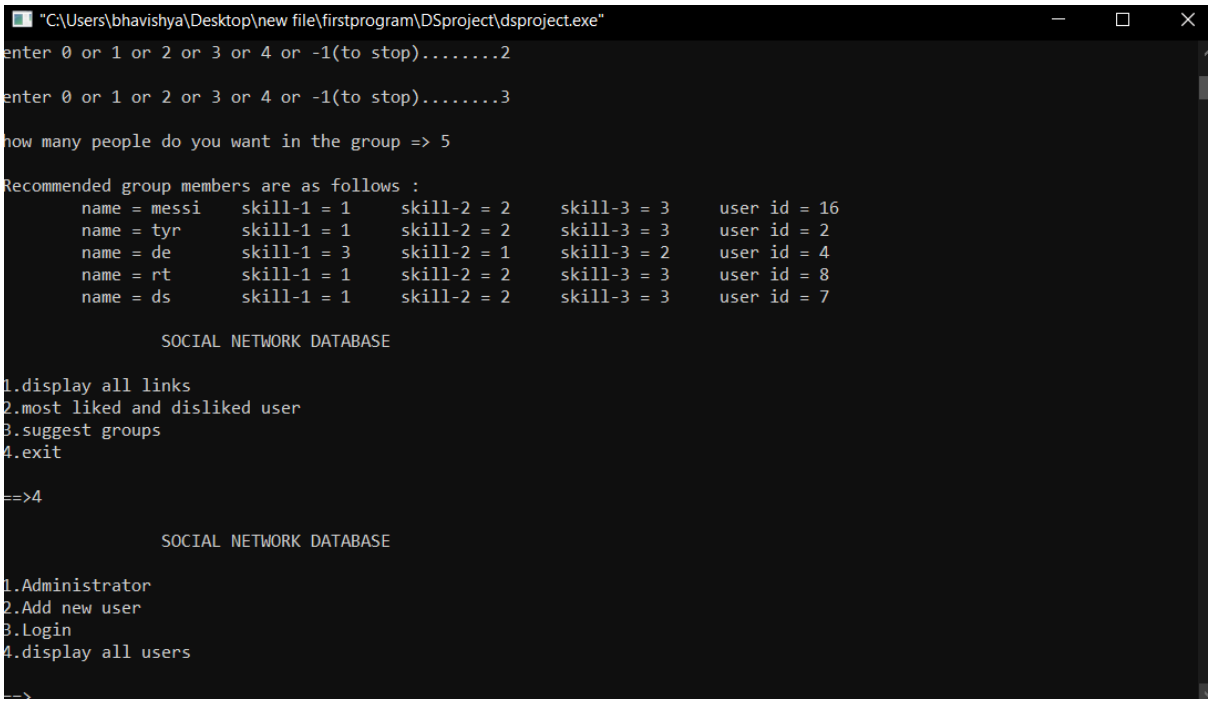
If user opt for second choice i.e. most liked users and disliked users then the user with maximum friends(most liked) and the user with minimum friends(most disliked) is displayed on the screen.



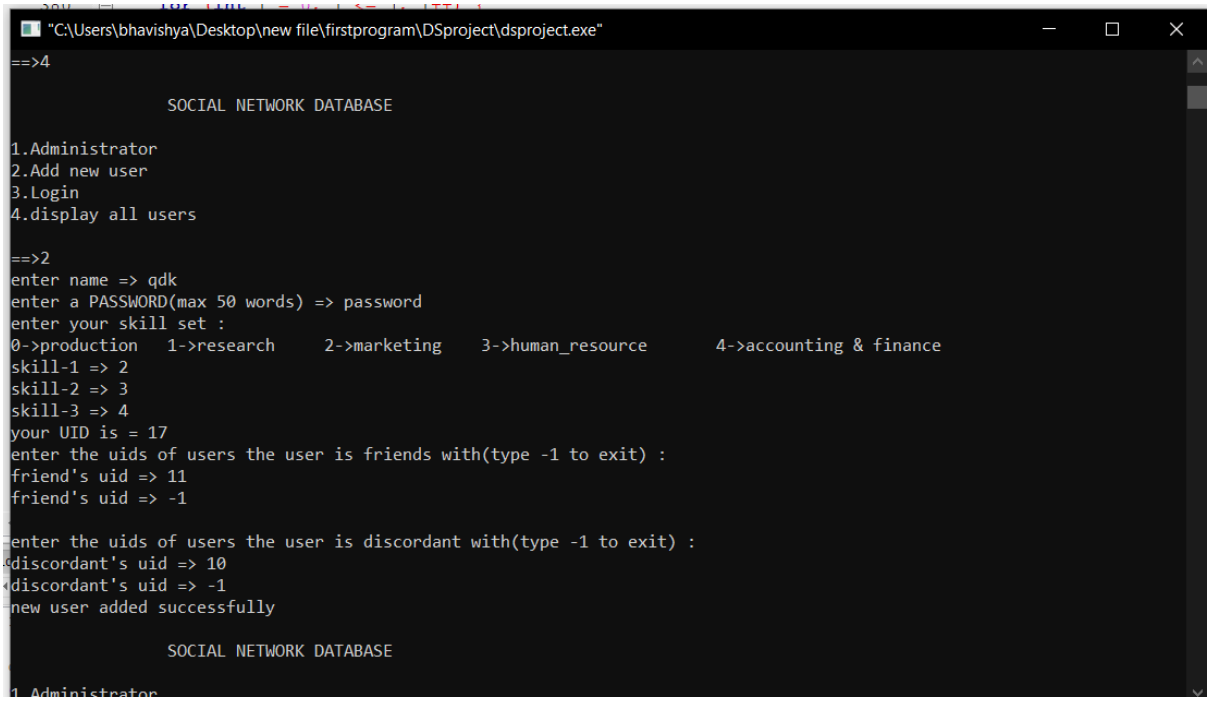
If user opt for last choice i.e. suggested groups then it asks the user to select the skill set of the group from the above given choices then after providing it to all the skill sets it asks for the number of people user want in a group, then it shows all the group member with the provided skill set that is suggest group members for the group.



After thar user opt for choice 4 that is end.

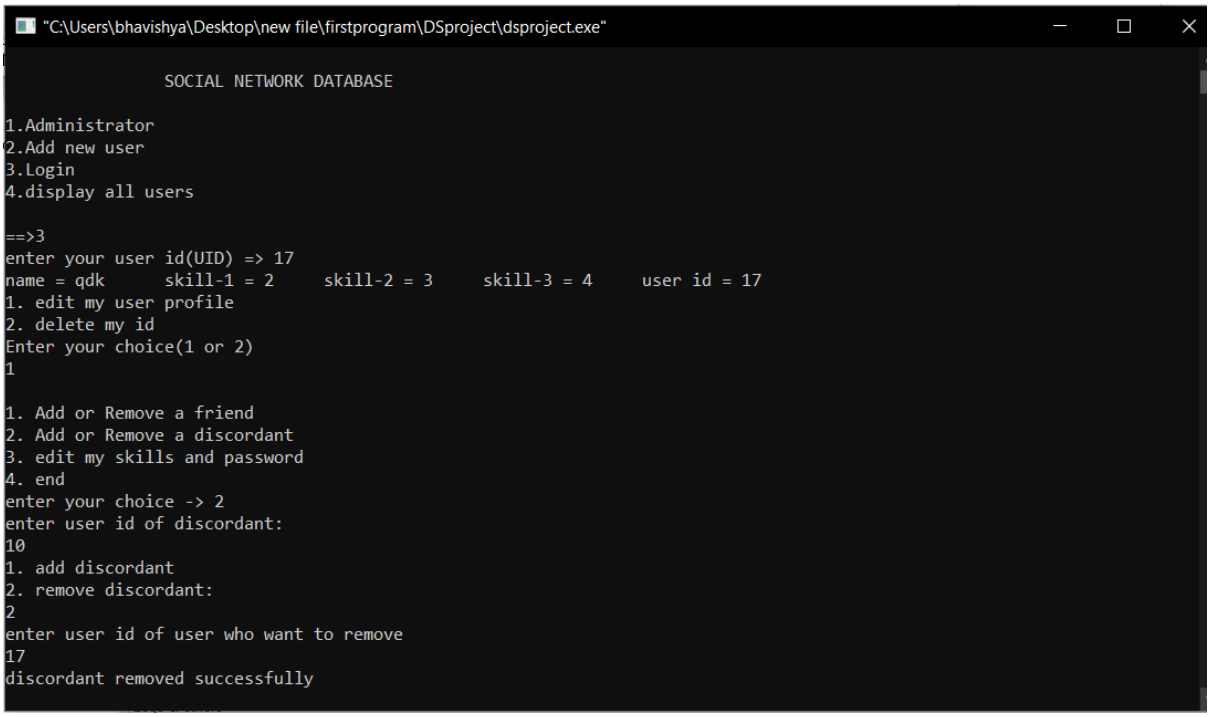


After that we will move to next module that is **2. Add new user** then it asks for user name, password that you are going to set for account, skill sets and then it allots user id to the user and asks for any user if the user want to make friend or discordant while creating account. The functions that are used for performing the tasks are given above in description section.



If user opt for choice **3. Login** then firstly it asks for user id then show all the user details and there are two choices

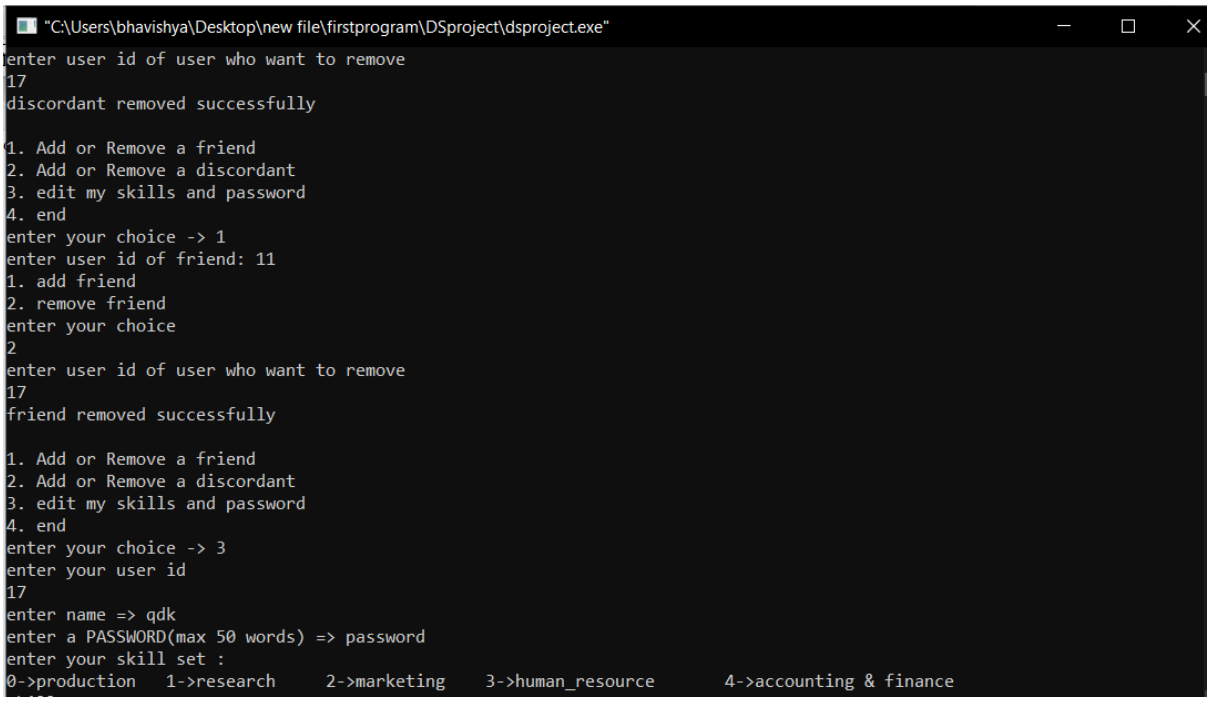
1. Edit my user profile
2. Delete my id



If user choose 1. Edit my user profile then there are three choices more

1. Add or remove friend
2. Add or remove discordant

3. Edit my user details
4. Exit



If user selects add or remove friend then it asks for

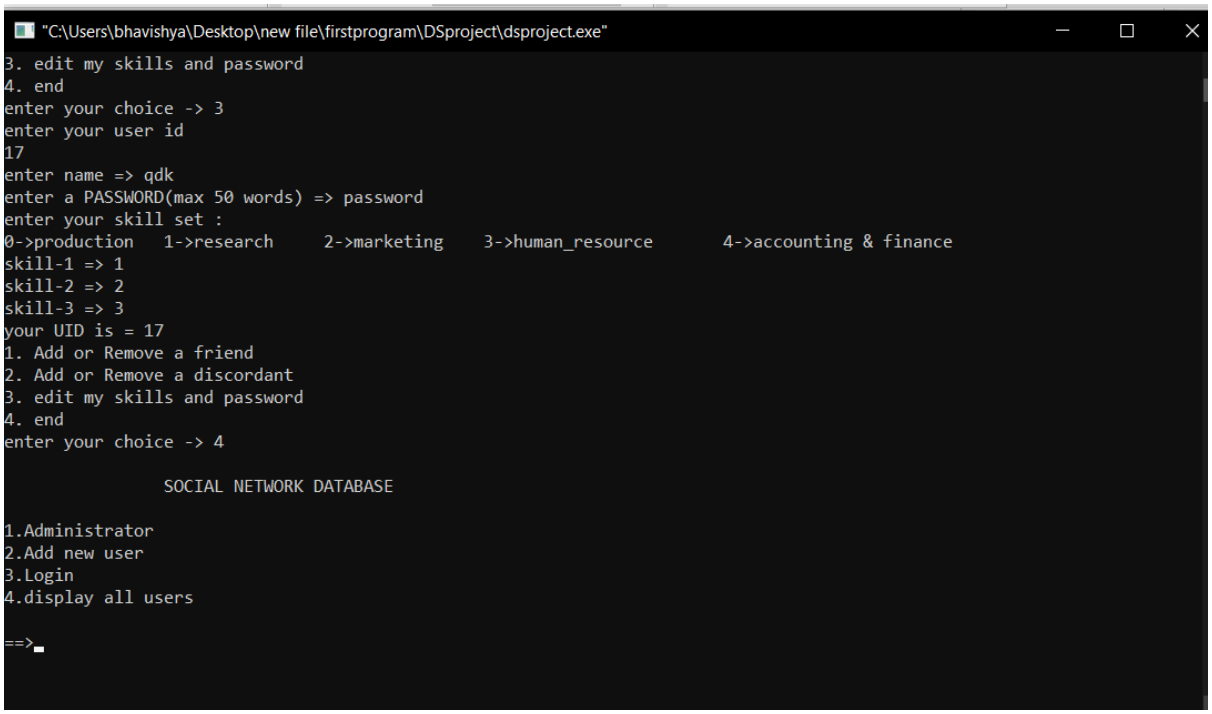
1. Add friend
2. Remove friend

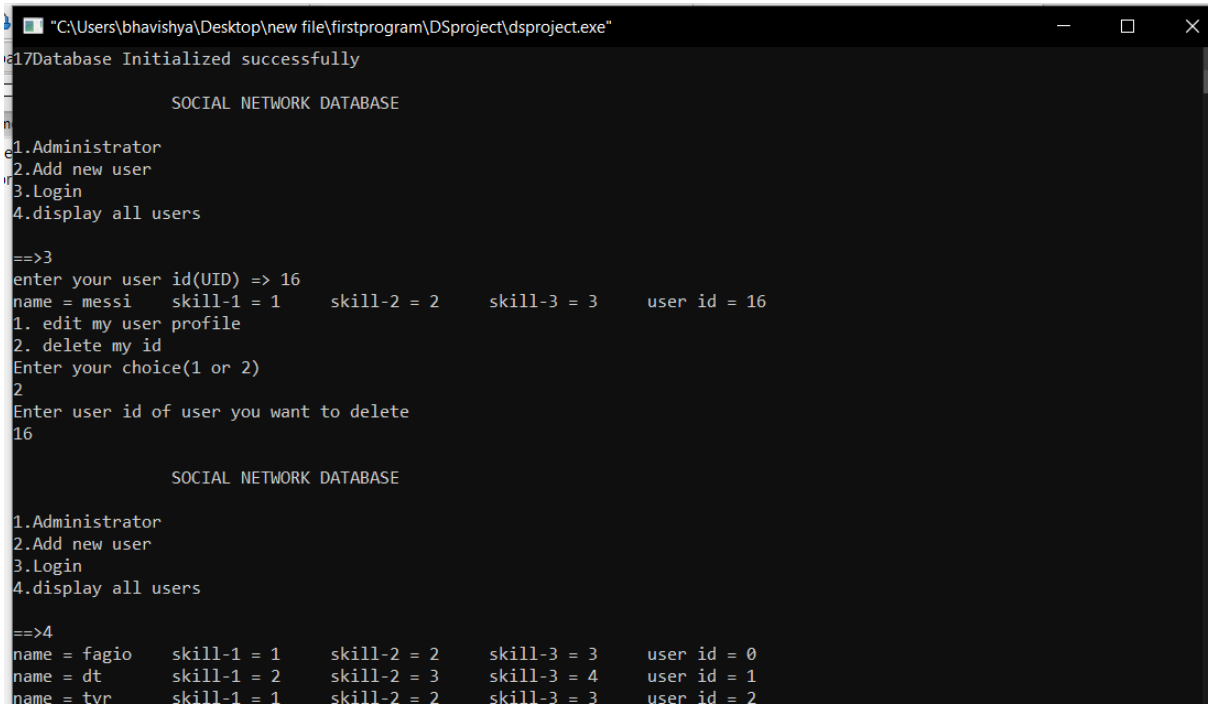
Then after selecting one by one user can easily add any friend and remove a friend.

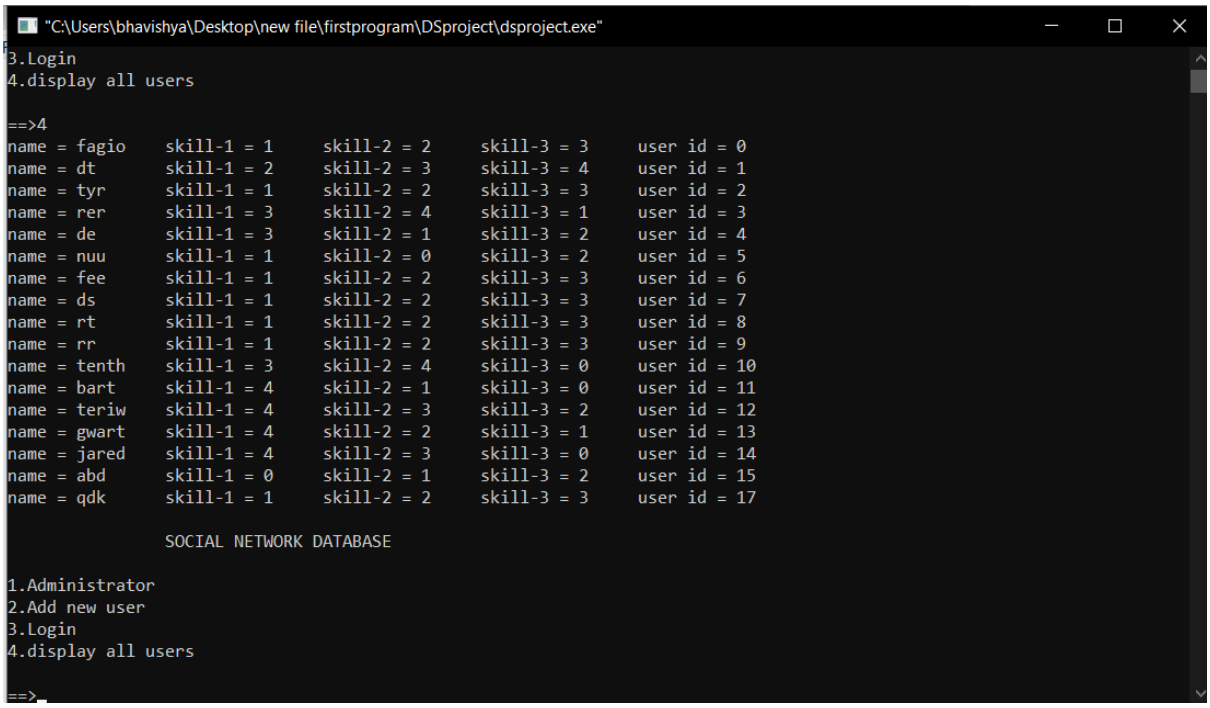
Similarly, user can also add and remove discordant by opting for that case.

The third one is edit my user details in this case user is allowed to change their password, skill set etc.

The fourth and the last is exit.


"C:\Users\bhavishya\Desktop\new file\firstprogram\DSproject\dsproject.exe"
3. edit my skills and password
4. end
enter your choice -> 3
enter your user id
17
enter name => qdk
enter a PASSWORD(max 50 words) => password
enter your skill set :
0->production 1->research 2->marketing 3->human_resource 4->accounting & finance
skill-1 => 1
skill-2 => 2
skill-3 => 3
your UID is = 17
1. Add or Remove a friend
2. Add or Remove a discordant
3. edit my skills and password
4. end
enter your choice -> 4
SOCIAL NETWORK DATABASE
1.Administrator
2.Add new user
3.Login
4.display all users
==>


"C:\Users\bhavishya\Desktop\new file\firstprogram\DSproject\dsproject.exe"
17Database Initialized successfully
SOCIAL NETWORK DATABASE
1.Administrator
2.Add new user
3.Login
4.display all users
==>3
enter your user id(UID) => 16
name = messi skill-1 = 1 skill-2 = 2 skill-3 = 3 user id = 16
1. edit my user profile
2. delete my id
Enter your choice(1 or 2)
2
Enter user id of user you want to delete
16
SOCIAL NETWORK DATABASE
1.Administrator
2.Add new user
3.Login
4.display all users
==>4
name = fagio skill-1 = 1 skill-2 = 2 skill-3 = 3 user id = 0
name = dt skill-1 = 2 skill-2 = 3 skill-3 = 4 user id = 1



It can be seen there user id 16 is deleted from database.

If user opt for last choice that display all users then all the users present in the Database are displayed on the screen.