Data Structures and Algorithms

Social Network Graph

Dated 2020-11-21

Question

Develop a social network website with thousands of users. With the use of data structure to efficiently represent and store the users' profiles. Keep the information about how these users are connected with each other.

Find out the friends for a given user. This search needs to be executed quickly as there are thousands of users.

Find the common friends for two given users. Propose a suitable data structure; implement the insert, delete, search operations and relevant algorithms to solve the problem. Analyse the time and space complexities of all the operations.

Solution

For the entire program visit: https://github.com/psymbio/social-network-graph

Initially, I thought of constructing two 2D matrices, one which would hold the information of all the users and the other that holds the data of how these people are connected with each other.

id	username	password	name	email
1	cornetto_tornado	severussnaps	Rick Roll	cornetto@yahoo.co.uk
2	xX-witchyPrincess-Xx	hhsj112@	Chandler Bing	
3	dancingsloths	nothing	Julian Assange	mumbo@jumbo.com

0	1	2	3	4
1	1	0	1	0
2	1	1	0	0
3	0	0	1	1

But with thousands of users, the CSV file of the connections crashed my text-editor. Not only that, but if there was a 25% chance someone was your friend then about 50% of the space used by the graph goes to waste, which is a great deal if you're handling thousands of users.

Then I decided to make an adjacency-list representation of a graph. This would only deal with the ids of each user in the graph to show the connection, saving space.

Now to create the graph:

```
// structure to represent an user node
struct user_node
{
    int dest;
    struct user_node *next;
};

// structure to represent the list of users
struct user_list
{
    struct user_node *head;
};

// graph is an array of adjacency lists.
```

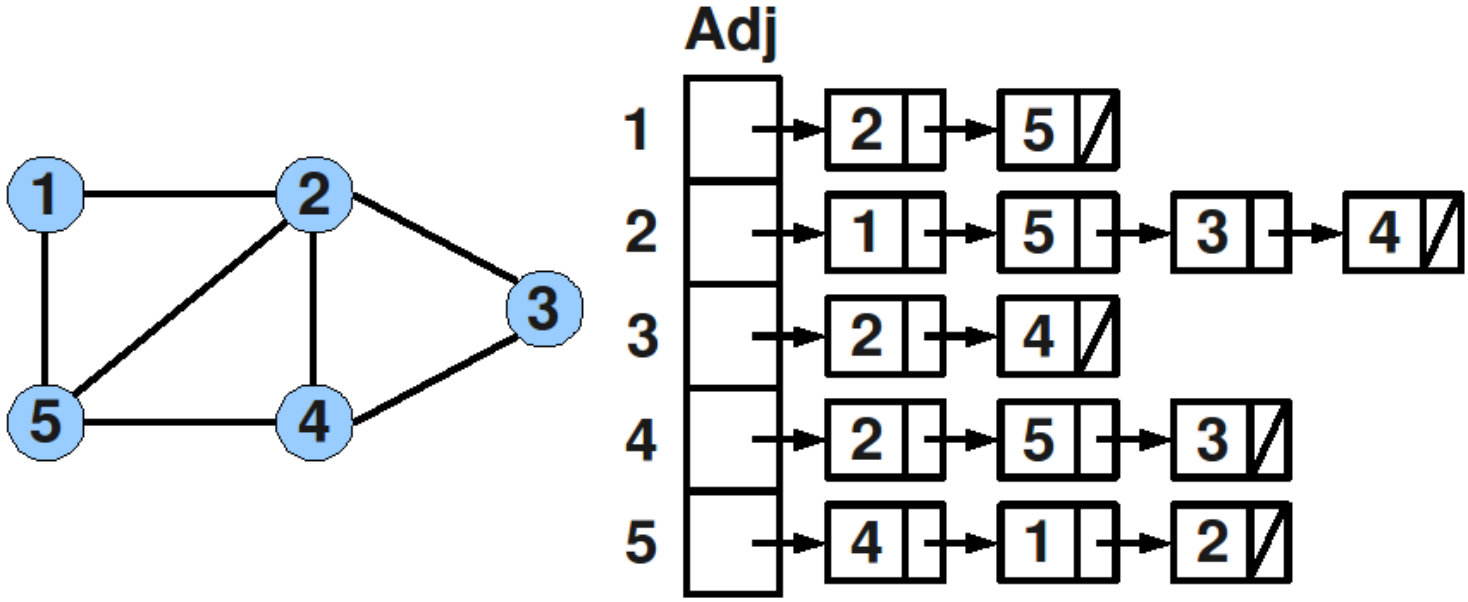


Figure 1: Adjacency Graph

```
struct Graph
{
    int V;
    struct user_list *array;
};

// A utility function to create a new adjacency list node
struct user_node *newuser_node(int dest)
{
    struct user_node *newNode = (struct user_node *)malloc(sizeof(struct user_node));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

// create a graph of V vertices
struct Graph *createGraph(int V)
{
    struct Graph *graph = (struct Graph *)malloc(sizeof(struct Graph));
    graph->V = V;
    graph->array = (struct user_list *)malloc(V * sizeof(struct user_list));
    int i;
    for (i = 0; i < V; ++i)
        graph->array[i].head = NULL;
    return graph;
}
```

Now that our graph has been implemented, let's carry out some functionality of the program.

1. Add a Friend

```
// add an edge to an undirected graph
void add_friend(struct Graph *graph, int src, int dest)
{
    // add edge from src to dest
    struct user_node *newNode = newuser_node(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;
```

```
    // add edge from dest to src
    newNode = newuser_node(src);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}
```

This simply adds an edge between two user nodes of the graph and takes linear time O(1).

2. Find a Node

```
int find_user(struct Graph *graph, int src)
{
    for (int v = 0; v < graph->V; ++v)
    {
        struct user_node *pCrawl = graph->array[v].head;
        while (pCrawl)
        {
            if (pCrawl->dest == src)
                return 1;
            pCrawl = pCrawl->next;
        }
    }
}
```

This takes O(n^2) time complexity to find a user_node.

3. Remove a Friend

```
void remove_friend(struct Graph *graph, int a, int b)
{
    if(find_user(graph, src) && find_user(graph, dest))
    {
        struct user_node *aCrawl = graph->array[v].head;
        struct user_node *bCrawl = graph->array[v].head;
        while (aCrawl || bCrawl)
        {
            if(aCrawl->dest == b)
                free(graph->array);
            aCrawl = aCrawl->next;
            if(bCrawl->dest == a)
                free(graph->array);
            bCrawl = bCrawl->next;
        }
    }
}
```

This takes O(n) time complexity.

4. Delete Your Account

```
void delete_user(struct Graph *graph, int a)
{
    if (find_user(graph, a))
    {
        for (int v = 0; v < graph->V; ++v)
        {
            struct user_node *aCrawl = graph->array[v].head;
            if (v == a)
            {
                free(graph->array);
            }
        }
    }
}
```

```
void delete_your_account(struct Graph *graph, MYSQL *conn)
{
    delete_user(graph, atoi(current_uid));
    sprintf(sql_statement, "DELETE FROM user_database where id=%d", atoi(current_uid));
    if (mysql_query(conn, sql_statement))
    {
        finish_with_error(conn);
    }
    printf("\nSo long, and thanks for all the fish.\n");
}
```

Deleting your account deletes the array of friends you had and your information in the database. Since find_user takes O(n^2) so does this.

5. Print Entire Graph

```
void print_friend_graph(struct Graph *graph)
{
    int v;
    for (v = 0; v < graph->V; ++v)
    {
        struct user_node *pCrawl = graph->array[v].head;
        printf("\nFriends of %d are:\n", v);
        while (pCrawl)
        {
            printf("-> %d", pCrawl->dest);
            pCrawl = pCrawl->next;
        }
        printf("\n");
    }
}
```

This prints the entire graph in O(n^2) however this is only a back-end function that normal users cannot run.

6. Print the Friends of a User

```
void print_userid_friends(struct Graph *graph, int v, MYSQL *conn, MYSQL_ROW row)
{
    int status = 0;
    struct user_node *pCrawl = graph->array[v].head;
    while (pCrawl)
    {
        sprintf(sql_statement, "SELECT username, first_name, last_name
        FROM user_database WHERE id=%d", pCrawl->dest);
        if (mysql_query(conn, sql_statement))
        {
            finish_with_error(conn);
        }
        MYSQL_RES *result = mysql_store_result(conn);
        while ((row = mysql_fetch_row(result)) != NULL)
        {
            printf("Username: %s\t", row[0]);
            printf("Name: %s %s\n", row[1], row[2]);
        }
        pCrawl = pCrawl->next;
    }
}
```

For a user with id = v this prints all the information of the friends of that user. This takes O(n) time.

7. Print Common Friends of Two Users

```
void find_common_friends(struct Graph *graph, int a, int b, MYSQL *conn, MYSQL_ROW row)
{
```

```
    printf("here 1");
    struct user_node *aCrawl = graph->array[a].head;
    struct user_node *bCrawl = graph->array[b].head;
    printf("here 1");
    while (aCrawl || bCrawl)
    {
        if (aCrawl->dest == bCrawl->dest)
        {
            sprintf(sql_statement, "SELECT username, first_name, last_name FROM
            user_database WHERE id=%d", aCrawl->dest);
            if (mysql_query(conn, sql_statement))
            {
                finish_with_error(conn);
            }
            MYSQL_RES *result = mysql_store_result(conn);
            while ((row = mysql_fetch_row(result)) != NULL)
            {
                printf("Username: %s\t", row[0]);
                printf("Name: %s %s\n", row[1], row[2]);
            }
        }
        aCrawl = aCrawl->next;
        bCrawl = bCrawl->next;
    }
}

void print_common_friends(struct Graph *graph, MYSQL *conn, MYSQL_ROW row)
{
    char user_a[50];
    char user_b[50];
    char user_a_uid[50];
    char user_b_uid[50];
    printf("Enter the first user: ");
    scanf("%s", user_a);
    printf("Enter the second user: ");
    scanf("%s", user_b);
    sprintf(sql_statement, "SELECT id FROM user_database
    WHERE username='%s' OR username='%s'", user_a, user_b);
    if (mysql_query(conn, sql_statement) != 0)
    {
        printf("Query failed  with the following message:\n");
        printf("%s\n", mysql_error(conn));
        exit(1);
    }
    MYSQL_RES *result = mysql_store_result(conn);
    while ((row = mysql_fetch_row(result)) != NULL)
    {
        strcpy(user_a_uid, row[0]);
        strcpy(user_b_uid, row[1]);
    }
    printf("%s", user_a_uid);
    printf("%s", user_b_uid);
    printf("The common friends of %s and %s are:\n", user_a, user_b);
    find_common_friends(graph, atoi(user_a_uid), atoi(user_b_uid), conn, row);
}
```

8. Print Friend Recommendations

```
void print_friend_recommendations(struct Graph *graph, int v, MYSQL *conn, MYSQL_ROW row)
{
    struct user_node *pCrawl = graph->array[v].head;
    while (pCrawl)
```

```
    {
        if (pCrawl->dest != v)
        {
            sprintf(sql_statement, "SELECT username, first_name, last_name FROM user_database WHERE id=%d", pCra
            if (mysql_query(conn, sql_statement))
            {
                finish_with_error(conn);
            }
            MYSQL_RES *result = mysql_store_result(conn);
            while ((row = mysql_fetch_row(result)) != NULL)
            {
                printf("Recommendation from username: %s\t", row[0]);
                printf("Name: %s %s\n", row[1], row[2]);
                print_userid_friends(graph, pCrawl->dest, conn, row);
                printf("\n\n\n");
            }
        }
        pCrawl = pCrawl->next;
    }
}
```

This takes O(n*k) where k is the number of friends user with id = v has.

9. Insert a New Login

```
int sign_up(MYSQL *conn, MYSQL_ROW row)
{
    char uid[50];
    if (mysql_query(conn, "SELECT id FROM user_database
    ORDER BY ID DESC LIMIT 1") != 0)
    {
        printf("Query failed  with the following message:\n");
        printf("%s\n", mysql_error(conn));
        exit(1);
    }
    MYSQL_RES *result = mysql_store_result(conn);
    while ((row = mysql_fetch_row(result)) != NULL)
    {
        strcpy(uid, row[0]);
    }
    char *new_uid = string_test(uid);
    char username[50], password[50], first_name[50], last_name[50], gender[50];
    char dob[50], phone_number[50], email[50], lives_in[50], college[50];
    printf("Enter your username: ");
    scanf("%s", username);
    printf("Enter your password: ");
    scanf("%s", password);
    printf("Enter your first name: ");
    scanf("%s", first_name);
    printf("Enter your last name: ");
    scanf("%s", last_name);
    printf("Enter your gender(Male/Female): ");
    scanf("%s", gender);
    printf("Enter your date of birth(YYYY-MM-DD): ");
    scanf("%s", dob);
    printf("Enter your mobile number(XXX-XXX-XXXX): ");
    scanf("%s", phone_number);
    printf("Enter your email: ");
    scanf("%s", email);
    printf("Enter your address: ");
    scanf("%s", lives_in);
    printf("Enter your college: ");
```

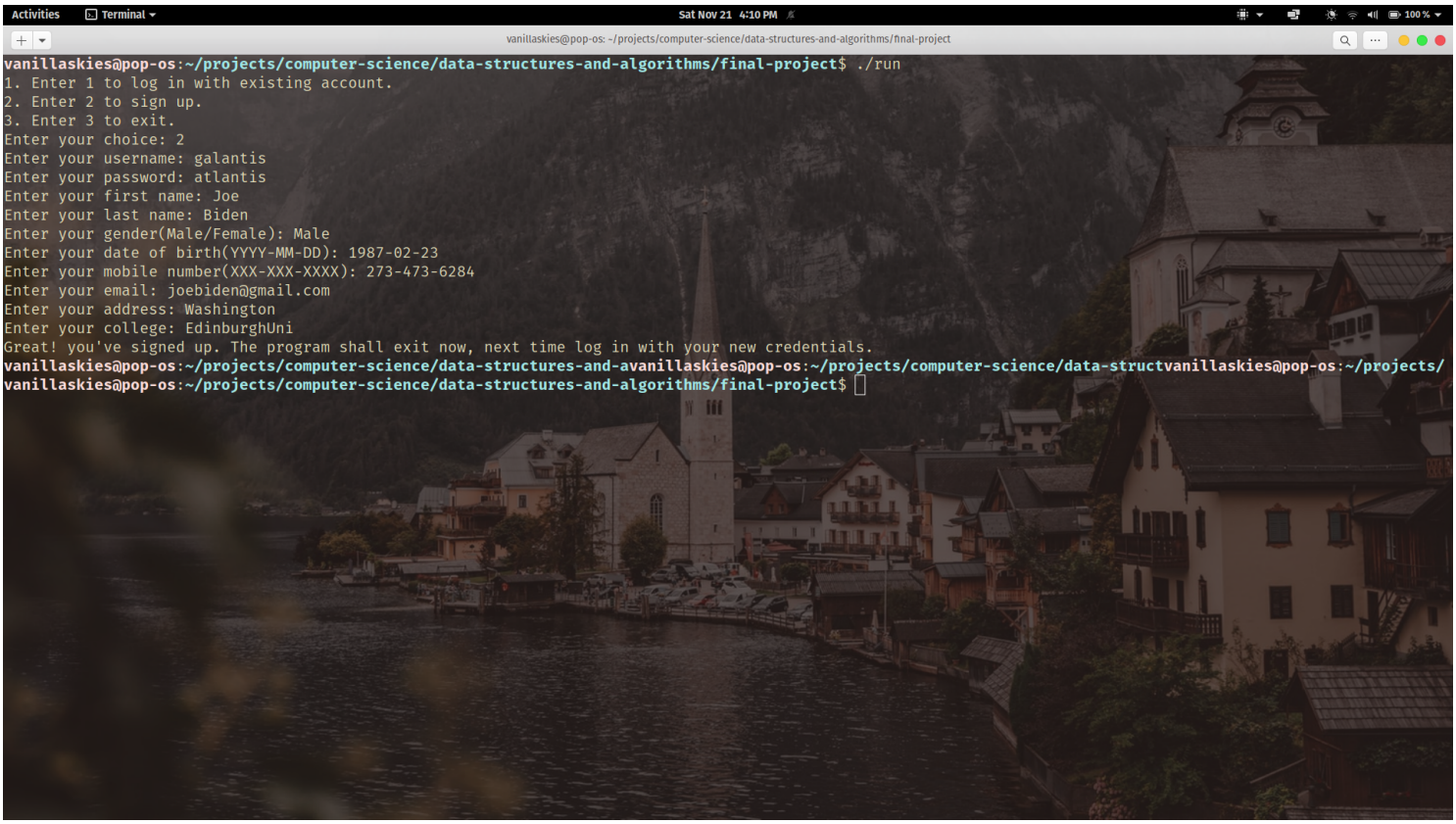
```
    scanf("%s", college);
    sprintf(sql_statement, "INSERT INTO user_database(id, username, password,
    first_name, last_name, gender, dob, since, phone_number,
    email, lives_in, college) VALUES('%s', '%s', '%s', '%s',
    '%s', '%s', '%s', '2020-11-29', '%s','%s', '%s', '%s')", new_uid,
    username, password,
    first_name, last_name, gender, dob, phone_number, email, lives_in, college);
    if (mysql_query(conn, sql_statement) != 0)
    {
        printf("Query failed  with the following message:\n");
        printf("%s\n", mysql_error(conn));
    }
    else
    {
        printf("Great! you've signed up. The program shall exit now,
        next time log in with your new credentials.\n");
        exit(0);
    }
}
```

This takes constant time.

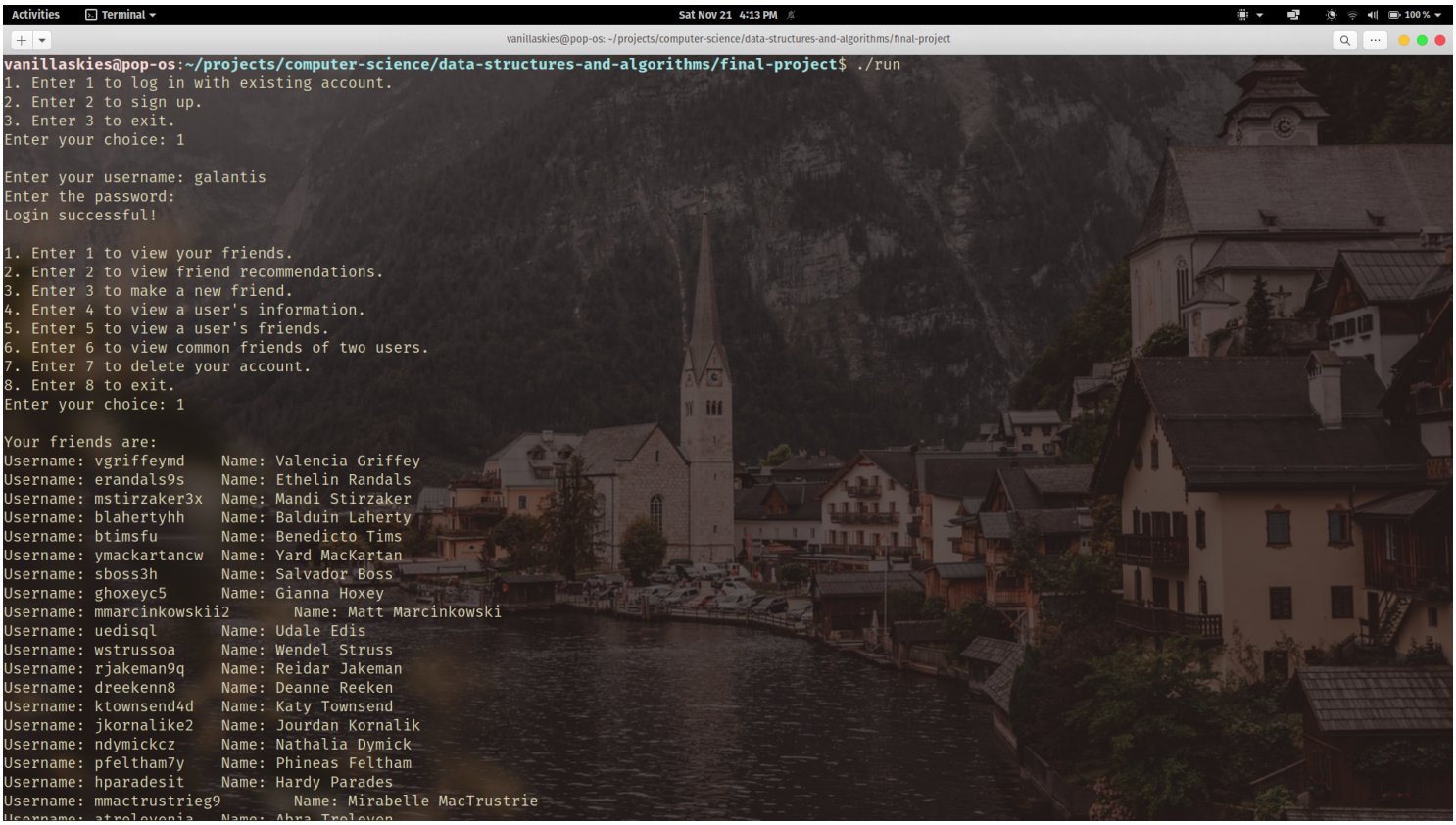
Output

For the entire program visit: https://github.com/psymbio/social-network-graph

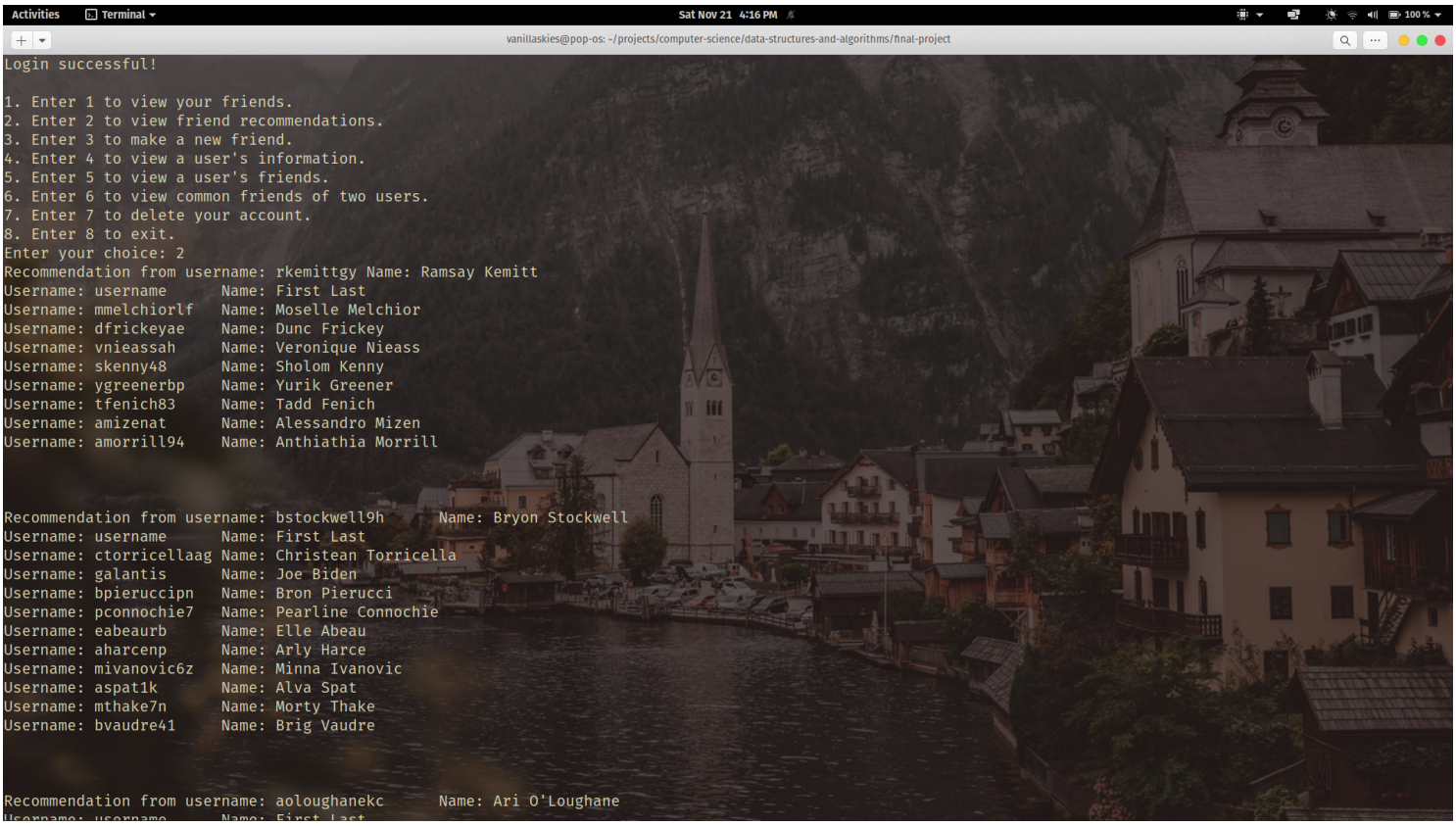
Create an account



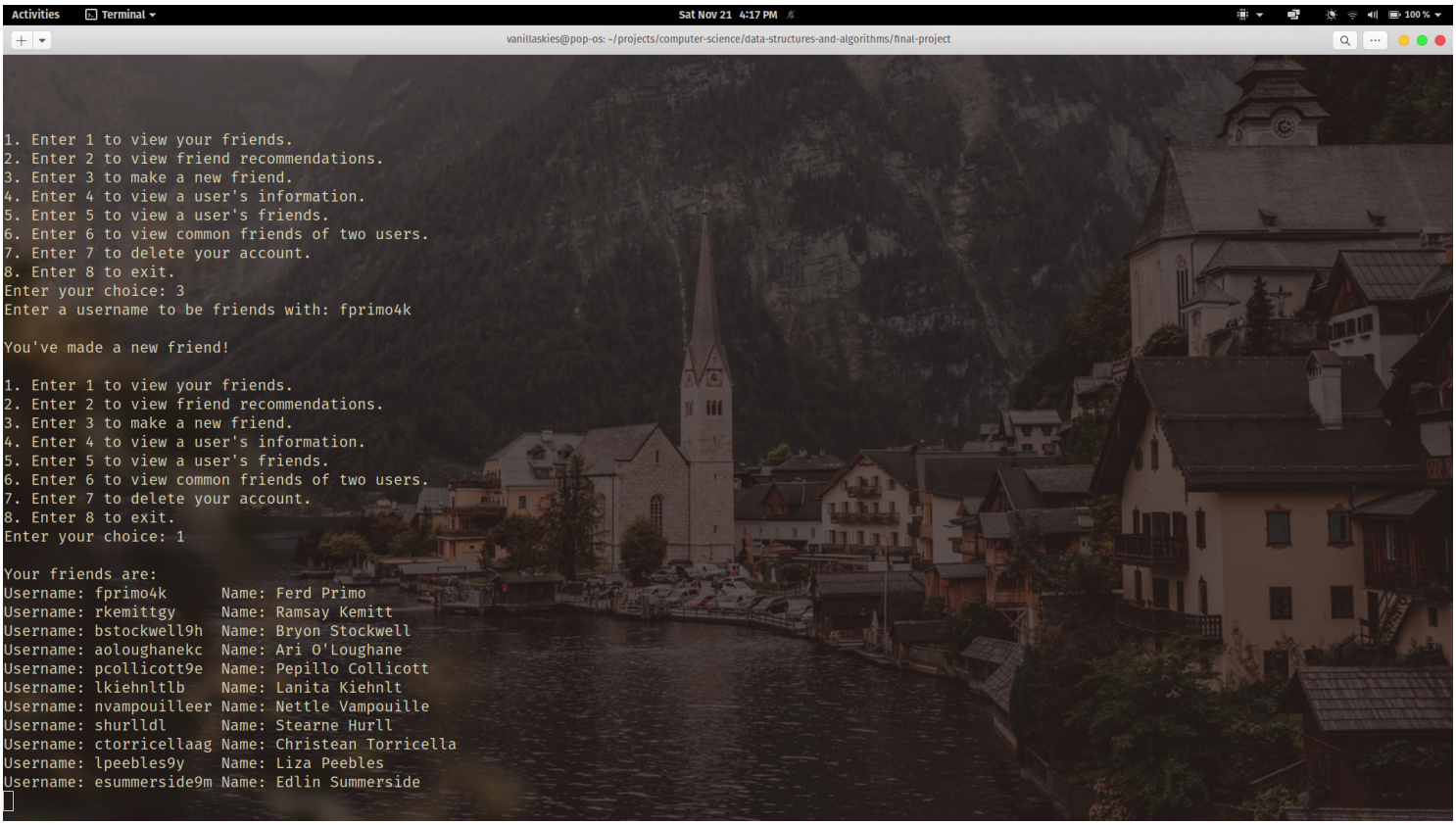
View Your Friends



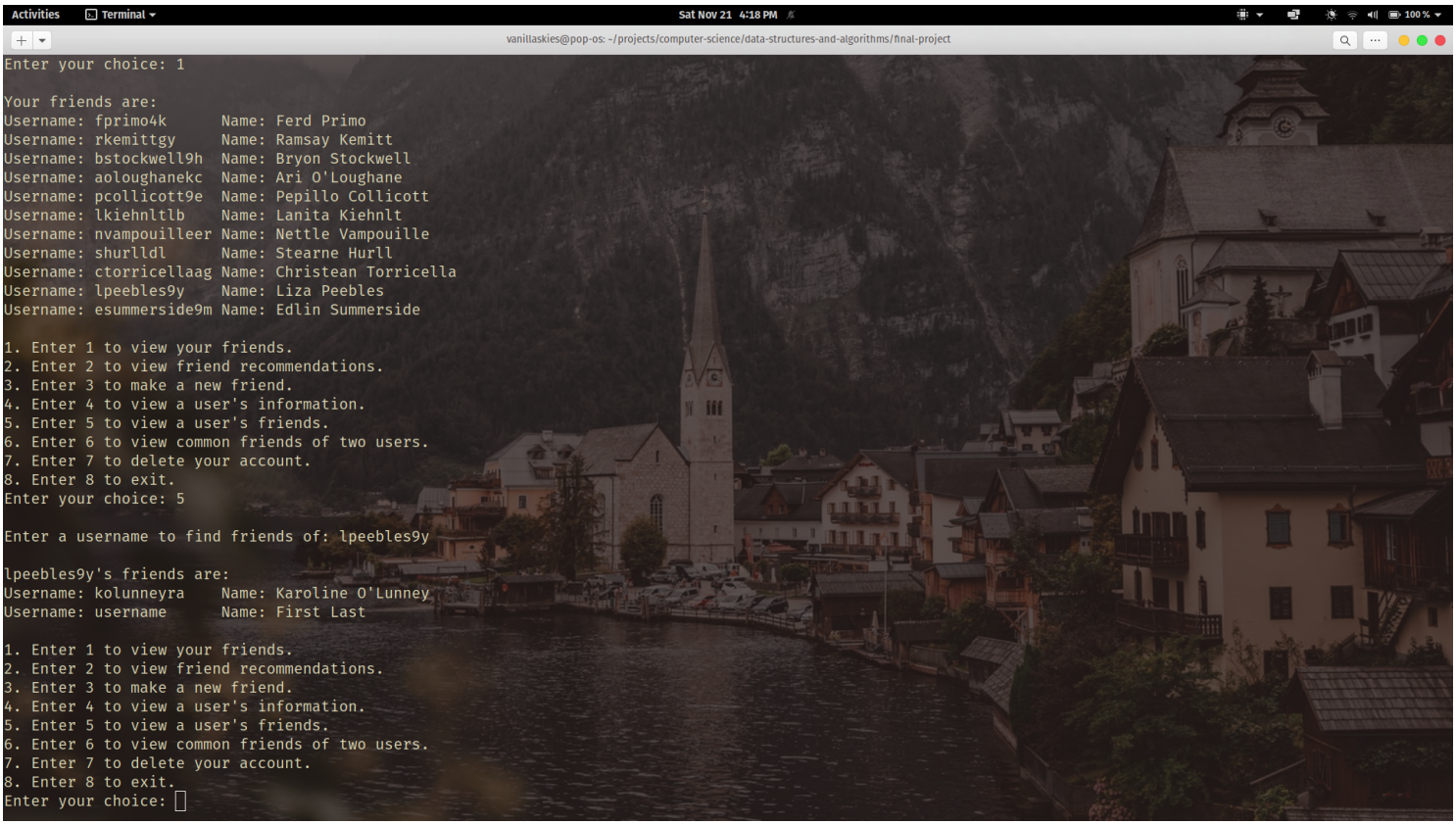
View Friend Recommendations



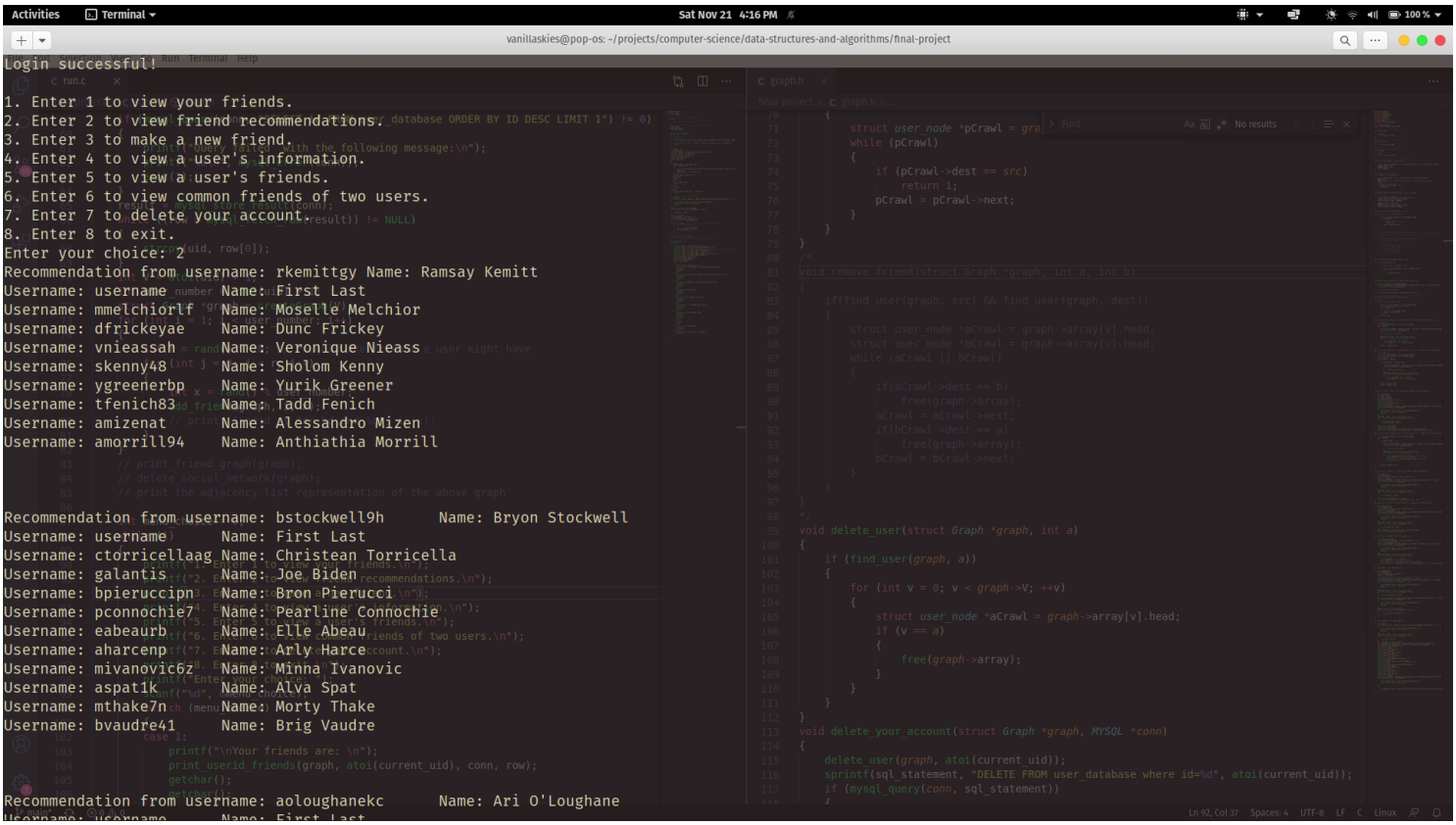
Make New Friends



View Another User's Information and Friends



View Common Friends



Delete Your Account