Jaypee Institute of Information Technology, Noida

Department of CSE & IT

Special Sem
(2021)

Mutual Friends in a Social Network

1) Title of the Project:

Mutual Friendships in a Social Network using BFS & DFS

2) Details of Team:

Aman Bhadouria - 19803020
Parish Bindal – 19803022
Nikhil Paleti - 19803024
Naivedya Khare - 19803002

3) Abstract of the project:

Aim of our mini-project is to provide solution and answer the below-mentioned questions intuitively through Text-to-Speech:

1. Finding the friends of all the people in the network.
2. Finding all the mutual friends for a given node in the network
3. Finding the shortest connection between two people in the network
4. Finding the n^{th} level friends for a person in the network

We can find the path between two people by running a BFS algorithm, starting the traversal from one person in level order until we reach the other person. Or in a much optimized way, we can run a bi-directional BFS from both the nodes until our search meets at some point and hence we conclude the path, meanwhile DFS being a depth wise traversal may run through many unnecessary sub-trees.

Moreover, in order to find friends at n^{th} level, using BFS this could be done in much less time, as this traversal keeps account of all the nodes in each level.

4) Tools and Technologies:

VS Code C++ IDE:

Visual Studio Code is a source-code editor made by Microsoft Features include support for

debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

SAPI:

SAPI is an API developed by Microsoft to allow the use of Speech Synthesis within Windows applications. To date, a number of versions of the API have been released, which have shipped either as part of a Speech SDK or as part of the Windows OS itself.

Windows API:

windows.h is a Windows-specific header file for the C and C++ Programming Languages, which contains declarations for all of the functions in the Windows API, all the common macros used by Windows programmers, and all the data types used by the various functions and subsystems. It defines a very large number of Windows specific functions that can be used in C/C++

5) Concepts Used:

Graph: A Graph is a Data Structure that has a finite set of nodes and vertices and a finite set of ordered pairs (u,v) which act as edges.

Graph Algorithms:

Floyd's Algorithm - In computer science, the Floyd–Warshall algorithm is an algorithm for finding shortest paths in a directed weighted graph. A single execution of the algorithm will find the lengths of shortest paths between all pairs of vertices.
Dijkstra's Algorithm - Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph
Prim and Kruskal's Algorithm - Prim and Kruskal Algorithms are greedy algorithms which finds a minimum spanning forest of a graph. If the graph is connected, it finds a minimum spanning tree.

Vectors:

A Vector is an STL-Defined implementation of Dynamic Array that allows resizing, as more data is input or removed.

Queue:

Queue is a data structure designed to operate in FIFO (First in First out) context. In queue elements are inserted from rear end and get removed from front end.

Classes:

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

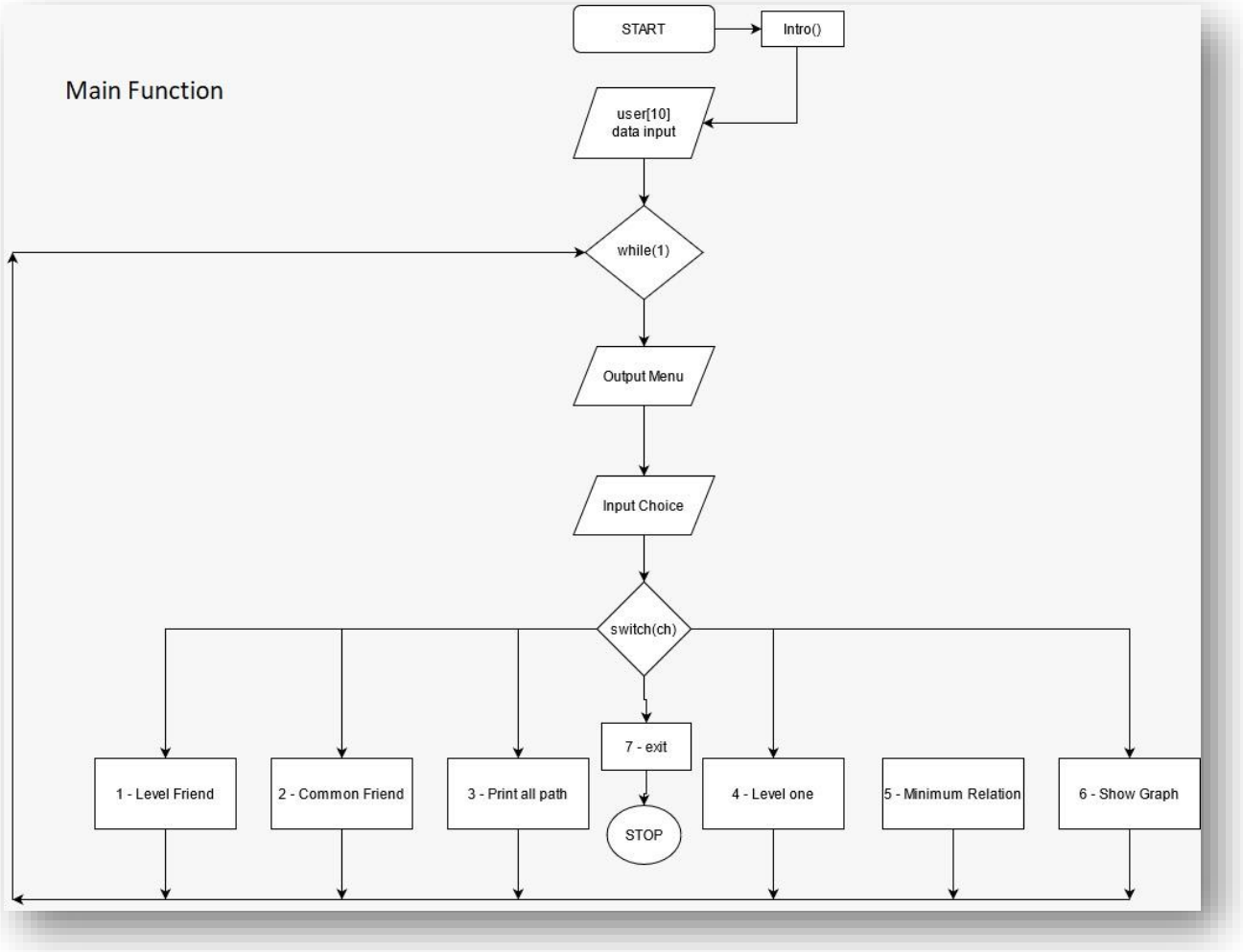
DFS (recursive):

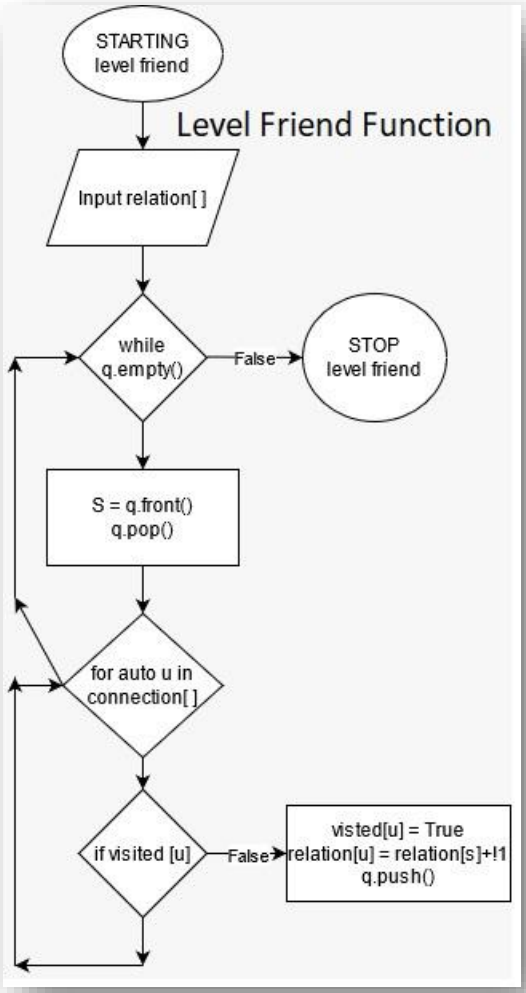
Depth–first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root for a graph) and explore as far as possible along each branch before backtracking.

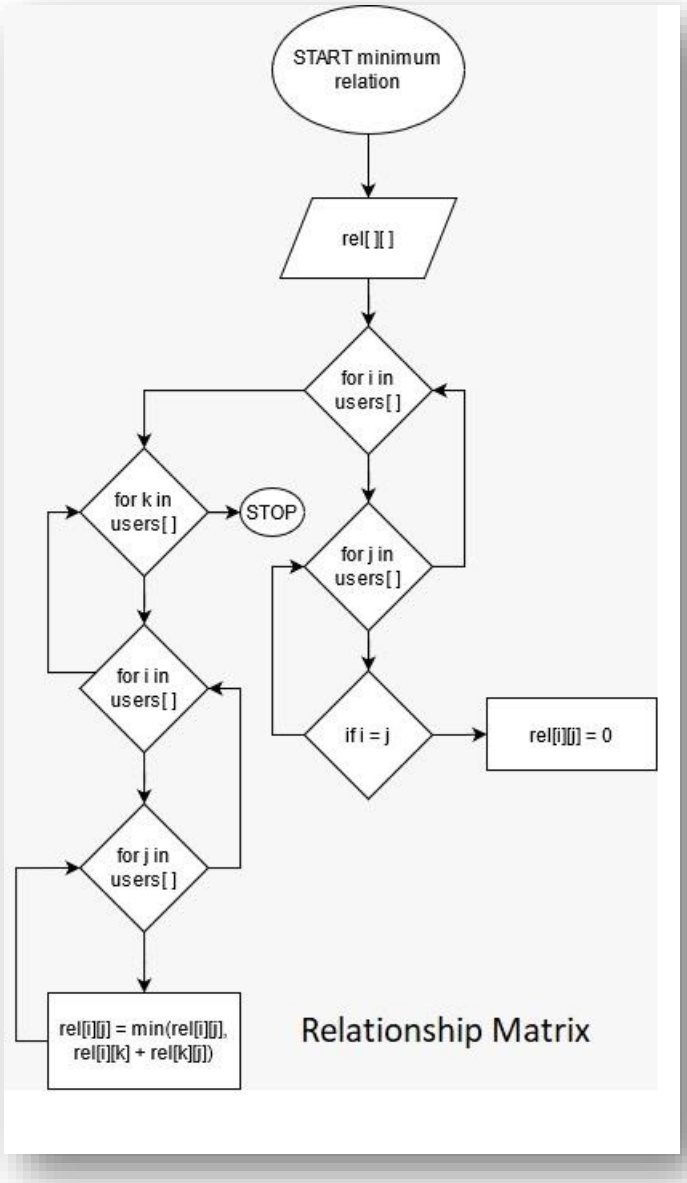
6) Dataset Description:

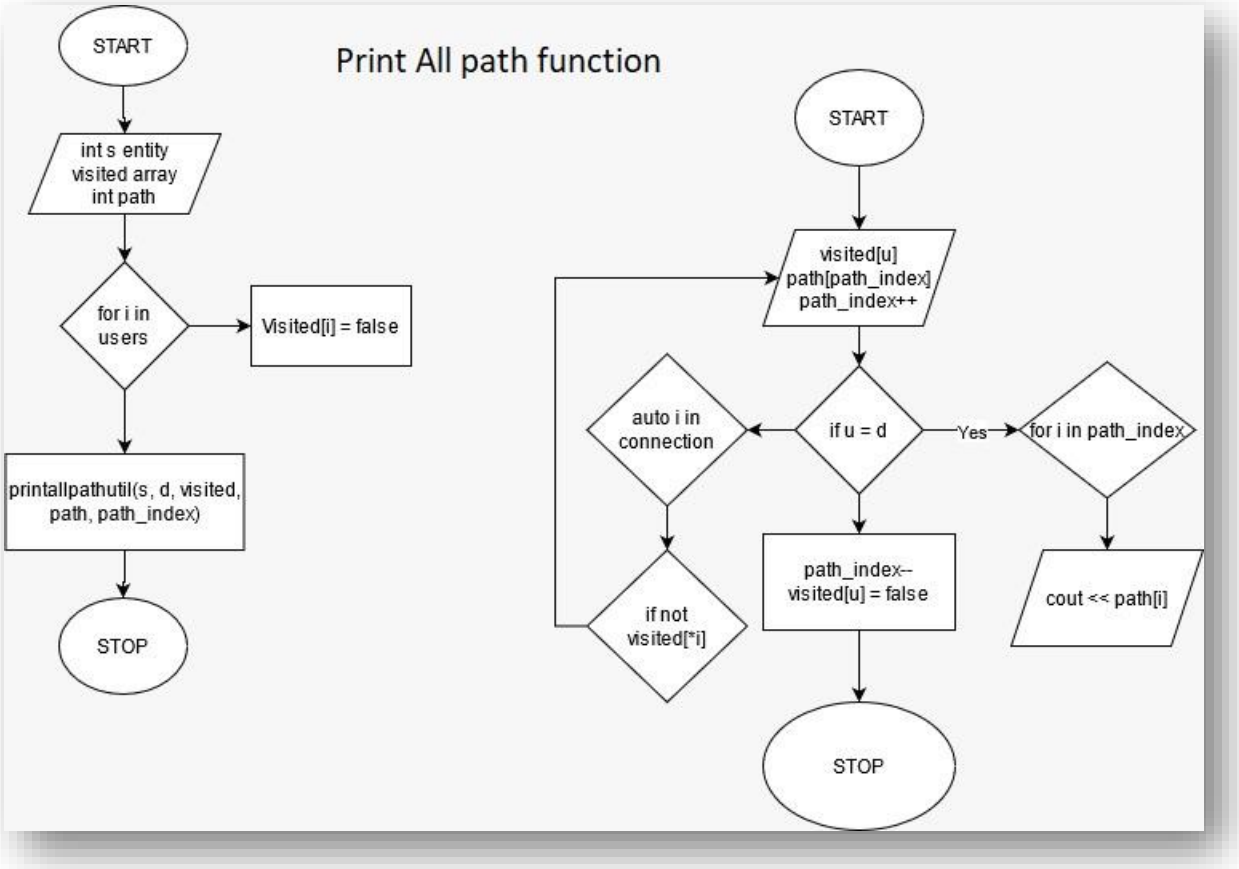
We have a Social Network of 10 students, implemented through Graph, who will be "connected" in the Social Network, as "Friends"

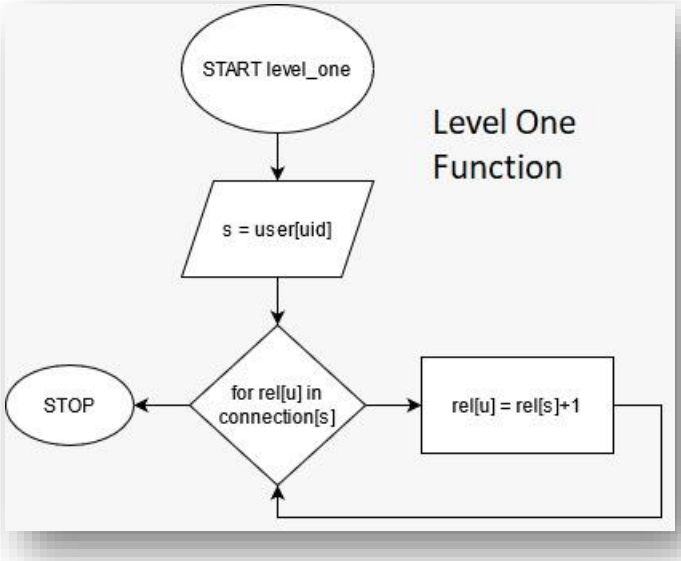
7) Project Design:




STARTING
level friend
Level Friend Function
Input relation[ ]
while
q.empty()
False
STOP
level friend
S = q.front()
q.pop()
for auto u in
connection[ ]
if visited [u]
False
visted[u] = True
relation[u] = relation[s]+!1
q.push()


START minimum
relation
rel[ ][ ]
for i in
users[ ]
for k in
users[ ]
STOP
for j in
users[ ]
for i in
users[ ]
if i = j
rel[i][j] = 0
for j in
users[ ]
rel[i][j] = min(rel[i][j],
rel[i][k] + rel[k][j])
Relationship Matrix


START
int s entity
visited array
int path
for i in
users
Visited[i] = false
printallpathutil(s, d, visited,
path, path_index)
STOP
Print All path function
START
visited[u]
path[path_index]
path_index++
auto i in
connection
if u = d
Yes
for i in path_index
if not
visited[*i]
path_index--
visited[u] = false
cout << path[i]
STOP


START level_one
Level One
Function
s = user[uid]
STOP
for rel[u] in
connection[s]
rel[u] = rel[s]+1

8) Implementation Details:

```
#include <bits/stdc++.h>
#include <windows.h>
#include <conio.h>
using namespace std;

#define users 10

void intro()
{
   cout << "Press enter when everyone is ready ";
   getch();
   system("cls");

   const char msgWelcome[] =
      "\t          ********************************************************\n\r"
      "\t          *                                      *\n\r"
      "\t          *       Welcome to Social Network Project          *\n\r"
      "\t          *                                      *\n\r"
      "\t          ********************************************************\n\r"
      "\t          *                                      *\n\r"
      "\t          *                                      *\n\r"
      "\t          *       Team Members : Aman Bhadouria              *\n\r"
      "\t          *                   Parish Bindal               *\n\r"
      "\t          *                   Nikhil Paleti             *\n\r"
      "\t          *                   Naivedya khare              *\n\r"
      "\t          *                                      *\n\r"
      "\t          ********************************************************\n";
   cout << msgWelcome;
   system("intro.vbs");
   getch();
   system("cls");
}

void image()
{
   system("photo.jpeg");
}
void red()
{
   HANDLE hConsole;
   hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hConsole, 4);
}
```

```
void white()
{
    HANDLE hConsole;
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hConsole, 7);
}
void t2s(char *x)
{
    ofstream batch;
    batch.open("working.vbs", ios::out);
    batch << "'\n";
    batch << "set speech = Wscript.CreateObject(\"SAPI.spVoice\") \n";
    batch << "Set speech.Voice = speech.GetVoices.Item(1)\n";
    batch << "Speech.Rate = 0.8\n";
    batch << "speech.speak\"" << x << "\" ";
    batch.close();

    system("working.vbs");
}

vector<int> connection[20];
int user_in = 0;

class user
{
    int uid;
    char name[20];
    int Nfriends;

public:
    user(char *names)
    {
        user_in++;

        strcpy(name, names);
        uid = user_in;
    }
    void level_friend(int relation[])
    {
        queue<int> q;
        bool visted[users + 1];
        for (int i = 0; i < users + 1; i++)
            visted[i] = false;
        visted[this->user::uid] = true;
        relation[this->user::uid] = 0;
```

```
        q.push(this->user::uid);

        while (!q.empty())
        {
            int s = q.front();
            q.pop();
            for (auto u : connection[s])
            {
                if (visted[u])
                    continue;
                visted[u] = true;
                relation[u] = relation[s] + 1;
                q.push(u);
            }
        }
        return;
    }
    void level_one(int relation[])
    {
        int s = this->user::uid;
        for (auto u : connection[s])
        {
            relation[u] = relation[s] + 1;
        }

        return;
    }

    char *retname()
    {
        return name;
    }
    int retuid()
    {
        return uid;
    }
};
void printcnt()
{

    for (int i = 1; i < users + 1; i++)
    {

        cout << "Connections of  "<< i << " are: ";
```

```
        for (auto it = connection[i].begin(); it != connection[i].end(); it++)
        {

            cout << *it << ' ';
        }
        cout << endl;
    }
}
void printmatrix(int mat[][users])
{
    cout << "\nABOVE RELATION WILL LOOK LIKE IN FORM OF MATRIX\n";
    cout << "   ";
    for (int i = 0; i < users; i++)
        cout << i + 1 << " ";
    for (int i = 0; i < users; i++)
    {
        cout << "\n";
        cout << i + 1 << "  ";
        for (int j = 0; j < users; j++)
            cout << mat[i][j] << " ";
        ;
    }
}
void minimumrelation(int mat[][users])
{
    int rell[users][users];
    for (int i = 0; i < users; i++)
    {
        for (int j = 0; j < users; j++)
        {
            if (i == j)
            {
                rell[i][j] = 0;
            }
            else if (mat[i][j] == 1)
            {
                rell[i][j] = mat[i][j];
            }
            else
                rell[i][j] = 10000;
        }
    }
    for (int k = 0; k < users; k++)
    {
        for (int i = 0; i < users; i++)
```

```
      {
         for (int j = 0; j < users; j++)
         {
            rell[i][j] = min(rell[i][j], rell[i][k] + rell[k][j]);
         }
      }
   }
   printmatrix(rell);
}

void printAllPathsUtil(int u, int d, bool visited[], int path[], int &path_index)
{
   visited[u] = true;
   path[path_index] = u;
   path_index++;

   if (u == d)
   {
      for (int i = 0; i < path_index; i++)
         cout << path[i] << " ";
      cout << endl;
   }
   else
   {

      for (auto i = connection[u].begin(); i != connection[u].end(); ++i)
         if (!visited[*i])
            printAllPathsUtil(*i, d, visited, path, path_index);
   }
   path_index--;
   visited[u] = false;
}
void printAllPaths(int s, int d)
{
   bool *visited = new bool[users + 1];
   int *path = new int[users + 1];
   int path_index = 0;
   for (int i = 0; i < users + 1; i++)
      visited[i] = false;
   printAllPathsUtil(s, d, visited, path, path_index);
}
void commonfriend(int s, int d, int relation[])
{
   for (auto it = connection[s].begin(); it != connection[s].end(); it++)
   {
```

```
        for (auto ti = connection[d].begin(); ti != connection[d].end(); ti++)
        {
            if (*it == *ti)
            {
                relation[*it] = 1;
            }
        }
    }
}

int main()
{
    int z;
    srand(time(0));
    z = rand() % 7 + 1;

    intro();

    user *account[users];
    cout << "The No. of users in network =" << users << "\n";
    int connection2[users][users];
    memset(connection2, 0, sizeof(connection2));

    char names[15][10] = {"Pairsh", "Rohit", "Aman", "Nikhil", "Divyansh", "Navi", "Aniket", "Shagun", "Sansk
ar", "Vikas"};
    vector<pair<int, int>> net = {{1, 2}, {1, 3}, {1, 6}, {2, 4}, {2, 6}, {3, 5}, {4, 8}, {5, 8}, {6, 10}, {7, 10}, {8, 9}, {1
0, 9}};
    for (int i = 0; i < users; i++)
    {
        account[i] = new user(names[i]);
    }
    int a, b;
    char ch = 'y';

    for (int i = 0; i < 12; i++)
    {
        a = net[i].first;
        b = net[i].second;
        connection2[a - 1][b - 1] = connection2[b - 1][a - 1] = 1;
        connection[a].push_back(b);
        connection[b].push_back(a);
    }
    printcnt();
    printmatrix(connection2);
```

```
int c;
while (1)
{
    cout << "\nEnter your choice\n";
    cout << "1.To obtain all level for friend connection\n";
    cout << "2.To Find all the mutual friends between two people in the network";
    red();
    cout << "(SAPI (Text-to-Speech Engine) Enabled)\n";
    white();
    cout << "3.To Find the connection links between two people in the network\n";
    cout << "4.To Find the friends of a person in the network";
    red();
    cout << "(SAPI (Text-to-Speech Engine) Enabled)\n";
    white();
    cout << "5.To Find minimum relation matrix\n";
    cout << "6. Open the graph depicting the above relations\n";
    cout << "7.EXIT\n";
    cin >> c;
    char Q[20];
    switch (c)
    {

    case 1:
        cout << "\n Enter the user id to obtain all level for friend connection ";
        cin >> a;
        int rel[users];
        account[a - 1]->level_friend(rel);
        for (int i = 1; i <= users; i++)
        {
            cout << i << " " << account[i - 1]->retname() << " --> " << rel[i] << "\n";
        }
        break;
    case 2:
        cout << "Enter the UID of users \n";
        cin >> a >> b;
        memset(rel, 0, sizeof(rel));
        commonfriend(a, b, rel);
        for (int i = 1; i <= users - 1; i++)
        {
            if (rel[i])
            {
                cout << i << " " << account[i - 1]->retname() << "\n";

                t2s(account[i - 1]->retname());
                t2s("is a Mutual Friend");
```

```
            }
        }

        break;
    case 3:
        cout << "Enter the UID of users \n";
        cin >> a >> b;
        printAllPaths(a, b);
        break;
    case 4:
        cout << "\n Enter the user id ";
        cin >> a;
        memset(rel, 0, sizeof(rel));
        account[a - 1]->level_one(rel);
        for (int i = 1; i <= users - 1; i++)
        {
            if (rel[i])
            {
                cout << i << " " << account[i - 1]->retname() << "\n";

                t2s(account[i - 1]->retname());
            }
        }
        t2s("are friends");
        break;
    case 5:
        minimumrelation(connection2);
        break;
    case 6:
        image();
        break;
    case 7:
        exit(1);
    default:
    {
        cout << "\nInvalid type! Did you mean option " << z << "?\n";
        srand(time(0));
        z = rand() % 7 + 1;
    }
    }
  }
  return 0;
}
```

9) References:

Microsoft SAPI (Documentation):
https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ee125663(v=vs.85)

Graphs (GeeksForGeeks):
https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

College PPTs