**Date: 17/01/2024**

**Roll No. and Name:** 21BCE049 Dev Bachani

**Course Code and Name:** 2CSDE56 Graph Theory

**AIM:** Use adjacency matrix and adjacency list for representing the graph. Use any of the representation to find union, intersection, complement, sum and difference of two graphs.

**Code:**

/\*

    Graph Operations Program

    This C++ program allows users to perform various operations on two undirected graphs,

    including finding the union, intersection, ring sum, and difference of the graphs.

    The program represents graphs using adjacency matrices and handles vertices that are absent

    in each graph. It provides a menu-driven interface for users to input graph details and

    perform desired operations.

    Author: Dev Bachani (21BCE049)

    Date: [10/01/2024]

    How to use:

    1. Enter the number of vertices and edges for each graph.

    2. Specify the vertices that are absent in each graph.

    3. Input edges for both graphs.

    4. View the adjacency matrices for both graphs.

    5. Obtain the union, intersection, ring sum, difference, and complement of the graphs.

    Note: Ensure that input values adhere to the program's expectations.

    Disclaimer: This program assumes that the input graphs are simple and undirected.

\*/

#include <bits/stdc++.h>

using namespace std;

#define ll long long int

#define vll vector<ll>

#define vint vector<int>

#define vpll vector<pair<ll, ll>>

#define vvll vector<vector<ll>>

#define pll pair<ll, ll>

#define mpll map<ll, ll>

#define ld long double

#define float double

const int num1 = 1e5 + 10;

vector<int> graph[num1];

bool vis[num1];

void dfs(int vertex)

{

    vis[vertex] = 1;

    cout << vertex << "->";

    for (int i = 0; i < graph[vertex].size(); i++)

    {

        int child = graph[vertex][i];

        if (vis[child] == 0)

            dfs(child);

    }

    // OR this for loop

    // for (int child : graph[vertex])

    // {

    //     if (vis[child] == 0)

    //         dfs(child);

    // }

}

// Function to find the intersection of two vectors

vector<int> intersection\_vectors(vector<int> &a, vector<int> &b)

{

    vector<int> intersection;

    int i = 0, j = 0;

    while (i < a.size() && j < b.size())

    {

        if (a[i] < b[j])

        {

            ++i;

        }

        else if (a[i] > b[j])

        {

            ++j;

        }

        else

        {

            // Found an intersection

            intersection.push\_back(a[i]);

            ++i;

            ++j;

        }

    }

    return intersection;

}

// Function to find the union of two vectors

vector<int> union\_vectors(vector<int> &a, vector<int> &b)

{

    vector<int> union1 = a;

    // int i = 0, j = 0;

    int j = 0;

    for (int i = 0; i < b.size(); i++)

    {

        if (union1[j] < b[i])

        {

            union1.push\_back(b[i]);

            // i++;

            j++;

        }

        if (union1[j] > b[i])

        {

            union1.push\_back(b[i]);

            j++;

        }

        else

        {

            j++;

            i++;

        }

    }

    sort(union1.begin(), union1.end());

    return union1;

}

// Function to print a matrix

void printMatrix(vector<vector<int>> &a)

{

    cout << "\n\n\n";

    cout << "Graph: \n\n";

    cout << "\tVertices\n";

    for (int i = 0; i < a.size(); i++)

    {

        cout << " " << i + 1 << "  ";

    }

    cout << "\n\n";

    for (auto row : a)

    {

        for (auto ele : row)

        {

            if (ele > -1)

                cout << " " << ele << "  ";

            else

                cout << ele << "  ";

        }

        cout << "\n";

    }

}

// Function to find the union of two matrices

vector<vector<int>> Union(vector<vector<int>> &a, vector<vector<int>> &b, vector<int> &a\_abs, vector<int> &b\_abs)

{

    cout << "Union\n\n";

    int v = a.size();

    vector<int> abs;

    set\_intersection(a\_abs.begin(), a\_abs.end(), b\_abs.begin(), b\_abs.end(), abs.begin());

    vector<vector<int>> G(v, vector<int>(v, 0));

    for (int i = 0; i < v; i++)

    {

        for (int j = 0; j < v; j++)

        {

            if (a[i][j] == 0 && b[i][j] == 0)

            {

                continue;

            }

            if (a[i][j] == 1 && b[i][j] == 1)

            {

                G[i][j] = 1;

                G[j][i] = 1;

            }

            if ((a[i][j] == 1 && b[i][j] == 0) || (a[i][j] == 0 && b[i][j] == 1))

            {

                G[i][j] = 1;

                G[j][i] = 1;

            }

            if ((a[i][j] == 1 && b[i][j] == -1) || (a[i][j] == -1 && b[i][j] == 1))

            {

                G[i][j] = 1;

                G[j][i] = 1;

            }

        }

    }

    return G;

}

// Function to find the intersection of two matrices

vector<vector<int>> Intersection(vector<vector<int>> &a, vector<vector<int>> &b, vector<int> &a\_abs, vector<int> &b\_abs)

{

    cout << "Intersection\n\n";

    int v = a.size();

    vector<int> abs = union\_vectors(a\_abs, b\_abs);

    vector<vector<int>> G(v, vector<int>(v, 0));

    for (int i = 0; i < v; i++)

    {

        for (int j = 0; j < v; j++)

        {

            if (a[i][j] == 0 && b[i][j] == 0)

            {

                continue;

            }

            if (a[i][j] == 1 && b[i][j] == 1)

            {

                G[i][j] = 1;

                G[j][i] = 1;

            }

        }

    }

    for (int i = 0; i < abs.size(); i++)

    {

        for (int j = 0; j < v; j++)

        {

            G[j][abs[i] - 1] = -1;

            G[abs[i] - 1][j] = -1;

        }

    }

    return G;

}

// Function to find the ring sum of two matrices

vector<vector<int>> Ring\_Sum(vector<vector<int>> &a, vector<vector<int>> &b, vector<int> &a\_abs, vector<int> &b\_abs)

{

    cout << "Ring Sum\n\n";

    vector<vector<int>> union\_graph = Union(a, b, a\_abs, b\_abs);

    vector<vector<int>> intersection\_graph = Intersection(a, b, a\_abs, b\_abs);

    int n = union\_graph.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (intersection\_graph[i][j] == 1)

            {

                union\_graph[i][j] = 0;

                union\_graph[j][i] = 0;

            }

        }

    }

    return union\_graph;

}

// Function to find the difference of two matrices

vector<vector<int>> Difference(vector<vector<int>> &a, vector<vector<int>> &b, vector<int> &a\_abs, vector<int> &b\_abs)

{

    cout << "Difference\n\n";

    vector<vector<int>> intersection\_graph = Intersection(a, b, a\_abs, b\_abs);

    vector<vector<int>> A = a;

    int n = intersection\_graph.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (intersection\_graph[i][j] == 1)

            {

                A[i][j] = 0;

                A[j][i] = 0;

            }

        }

    }

    return A;

}

// Function to find the complement of a matrix

vector<vector<int>> Complement(vector<vector<int>> &a)

{

    cout << "Complement\n\n";

    vector<vector<int>> A = a;

    int n = a.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (a[i][j] == 1)

            {

                A[i][j] = 0;

                A[j][i] = 0;

            }

            else if (a[i][j] == 0 && i != j)

            {

                A[i][j] = 1;

                A[j][i] = 1;

            }

        }

    }

    return A;

}

int main()

{

    int v1, e1, v2, e2;

    cout << "Enter number of vertices in graph 1: " << endl;

    cin >> v1;

    cout << "Enter number of vertices in graph 2: " << endl;

    cin >> v2;

    cout << "Enter number of edges in graph 1: " << endl;

    cin >> e1;

    cout << "Enter number of edges in graph 2: " << endl;

    cin >> e2;

    int v1\_, v2\_;

    int v = max(v1, v2);

    // cout<<v<<endl;

    cout << "Enter number of vertices absent in graph 1: " << endl;

    cin >> v1\_;

    cout << "Enter number of vertices absent in graph 2: " << endl;

    cin >> v2\_;

    vector<int> abs\_vertices\_1;

    vector<int> abs\_vertices\_2;

    for (int i = 0; i < v1\_; i++)

    {

        int a;

        cout << "Absent vertex of Graph 1: " << endl;

        cin >> a;

        abs\_vertices\_1.push\_back(a);

    }

    for (int i = 0; i < v2\_; i++)

    {

        int a;

        cout << "Absent vertex of Graph 2: " << endl;

        cin >> a;

        abs\_vertices\_2.push\_back(a);

    }

    sort(abs\_vertices\_1.begin(), abs\_vertices\_1.end());

    sort(abs\_vertices\_2.begin(), abs\_vertices\_2.end());

    vector<vector<int>> G1(v, vector<int>(v, 0));

    vector<vector<int>> G2(v, vector<int>(v, 0));

    // for (int i = 0; i < v1\_; i++)

    // {

    //     for (int j = 0; j < v; j++)

    //     {

    //         for (int k = 0; k < v; k++)

    //         {

    //             if (j == abs\_vertices\_1[i] - 1 || k == abs\_vertices\_1[i] - 1)

    //             {

    //                 G1[j][k] = -1;

    //             }

    //         }

    //     }

    // }

    // for (int i = 0; i < v2\_; i++)

    // {

    //     for (int j = 0; j < v; j++)

    //     {

    //         for (int k = 0; k < v; k++)

    //         {

    //             if (j == abs\_vertices\_2[i] - 1 || k == abs\_vertices\_2[i] - 1)

    //             {

    //                 G2[j][k] = -1;

    //             }

    //         }

    //     }

    // }

    // less complexity

    for (int i = 0; i < v1\_; i++)

    {

        for (int j = 0; j < v; j++)

        {

            G1[j][abs\_vertices\_1[i] - 1] = -1;

            G1[abs\_vertices\_1[i] - 1][j] = -1;

        }

    }

    for (int i = 0; i < v2\_; i++)

    {

        for (int j = 0; j < v; j++)

        {

            G2[j][abs\_vertices\_2[i] - 1] = -1;

            G2[abs\_vertices\_2[i] - 1][j] = -1;

        }

    }

    cout << "Enter Edges of Graph 1: \n";

    for (int i = 0; i < e1; i++)

    {

        int a, b;

        cout << "Enter the 2 vertices of edge " << i + 1 << endl;

        cin >> a >> b;

        G1[a - 1][b - 1] = 1;

        G1[b - 1][a - 1] = 1;

    }

    cout << "Enter Edges of Graph 2: \n";

    for (int i = 0; i < e1; i++)

    {

        int a, b;

        cout << "Enter the 2 vertices of edge " << i + 1 << endl;

        cin >> a >> b;

        G2[a - 1][b - 1] = 1;

        G2[b - 1][a - 1] = 1;

    }

    printMatrix(G1);

    cout << endl;

    printMatrix(G2);

    vector<vector<int>> G3 = Union(G1, G2, abs\_vertices\_1, abs\_vertices\_2);

    cout << endl;

    printMatrix(G3);

    vector<vector<int>> G4 = Intersection(G1, G2, abs\_vertices\_1, abs\_vertices\_2);

    cout << endl;

    printMatrix(G4);

    vector<vector<int>> G5 = Ring\_Sum(G1, G2, abs\_vertices\_1, abs\_vertices\_2);

    cout << endl;

    printMatrix(G5);

    vector<vector<int>> G6 = Difference(G1, G2, abs\_vertices\_1, abs\_vertices\_2);

    cout << endl;

    printMatrix(G6);

    vector<vector<int>> G7 = Complement(G1);

    cout << endl;

    printMatrix(G7);

}