Date: 10/1/2024
Roll No. and Name: 22BCE510 (Aarshit Jolapara)
Course Code and Name: 2CSDE56 – Graph Theory
Practical No.: 1
AIM:
1) Use the adjacency matrix and/or adjacency list for representing the graph. Use any of the representations to find the union, intersection, ring sum, and difference of two graphs.

2) Generate the nxn maze and generate the path from start point to end point.

AIM 1 using Adjacency Matrix::

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

void print(vector<vector<int>> adj) {
    cout << "Adjacency Matrix: " << endl;
    for(int i=1;i<adj.size();i++) {
        for(int j=1;j<adj[i].size();j++) {
            cout<<adj[i][j]<<" ";
        }
        cout << endl;
    }
    cout << endl;
}

vector<vector<int>> graphCreation(int n,int e) {
    vector<vector<int>> adj(n+1,vector<int>(n+1,0));
    for(int i=1;i<=e;i++) {
        int u,v;
        cin>>u>>v;
        adj[u][v]=1;
        adj[v][u]=1;
    }
    return adj;
}

// union of two graphs
vector<vector<int>> unionGraph(vector<vector<int>> adj1,vector<vector<int>>
adj2) {
```

```
    int s1 = adj1.size();
    int s2 = adj2.size();
    int r;

    vector<vector<int>> uni;
    if(s1 > s2) {
        uni = adj1;
        r = s2;
    }
    else {
        uni = adj2;
        r = s1;
    }

    for(int i=1;i<r;i++) {
        for(int j=1;j<r;j++) {
            if(adj1[i][j]==1 || adj2[i][j]==1) {
                uni[i][j]=1;
            }
        }
    }

    return uni;
}

// intersection of two graphs
vector<vector<int>> interGraph(vector<vector<int>> adj1,vector<vector<int>>
adj2) {

    int s1 = adj1.size();
    int s2 = adj2.size();
    int r;

    vector<vector<int>> inter;
    if(s1 > s2) {
        inter = vector<vector<int>>(s1,vector<int>(s1,0));
        r = s2;
    }
    else {
        inter = vector<vector<int>>(s2,vector<int>(s2,0));
        r = s1;
    }
```

```
    for(int i=1;i<r;i++) {
        for(int j=1;j<r;j++) {
            if(adj1[i][j]==1 && adj2[i][j]==1) {
                inter[i][j]=1;
            }
        }
    }

    return inter;
}

// difference of two graphs
vector<vector<int>> diffGraph(vector<vector<int>> adj1,vector<vector<int>>
adj2) {

    int s1 = adj1.size();
    int s2 = adj2.size();
    int r = min(s1,s2);

    vector<vector<int>> diff;
    diff = adj1;

    for(int i=1;i<r;i++) {
        for(int j=1;j<r;j++) {
            if(adj1[i][j]==1 && adj2[i][j]==1) {
                diff[i][j]=0;
            }
        }
    }

    return diff;
}

// symmetric difference of two graphs
vector<vector<int>> symDiffGraph(vector<vector<int>> adj1,vector<vector<int>>
adj2) {
    vector<vector<int>> un = unionGraph(adj1,adj2);
    vector<vector<int>> in = interGraph(adj1,adj2);
    return diffGraph(un,in);
}
```

```cpp
int main() {
    int n1,e1;
    cout << "Enter number of vertices and edges (Graph 1): ";
    cin>>n1>>e1;
    vector<vector<int>> g1 = graphCreation(n1,e1);
    print(g1);

    int n2,e2;
    cout << "Enter number of vertices and edges (Graph 2): ";
    cin>>n2>>e2;
    vector<vector<int>> g2 = graphCreation(n2,e2);
    print(g2);

    // union
    vector<vector<int>> ug = unionGraph(g1,g2);
    cout << "Union Graph :: " << endl;
    print(ug);

    // intersection
    vector<vector<int>> ig = interGraph(g1,g2);
    cout << "Intersection Graph :: " << endl;
    print(ig);

    // difference
    vector<vector<int>> dg = diffGraph(g1,g2);
    cout << "Difference Graph (g1-g2):: " << endl;
    print(dg);

    dg = diffGraph(g2,g1);
    cout << "Difference Graph (g2-g1):: " << endl;
    print(dg);

    // symmetric difference
    vector<vector<int>> sdg = symDiffGraph(g1,g2);
    cout << "Symmetric Difference Graph (Ring Sum) :: " << endl;
    print(sdg);

    return 0;
}
```

Input:

5 6
1 2
1 3
2 3
2 4
3 4
4 5

5 5
1 2
4 3
5 3
2 5
3 1

Output:

```
Enter number of vertices and edges (Graph 1): 5 6
1 2
1 3
2 3
2 4
3 4
4 5
Adjacency Matrix:
0 1 1 0 0
1 0 1 1 0
1 1 0 1 0
0 1 1 0 1
0 0 0 1 0

Enter number of vertices and edges (Graph 2):
5 5
1 2
4 3
5 3
2 5
3 1
Adjacency Matrix:
0 1 1 0 0
1 0 0 0 1
1 0 0 1 1
0 0 1 0 0
0 1 1 0 0

Union Graph ::
Adjacency Matrix:
0 1 1 0 0
1 0 1 1 1
1 1 0 1 1
0 1 1 0 1
0 1 1 1 0
```

```
Intersection Graph ::
Adjacency Matrix:
0 1 1 0 0
1 0 0 0 0
1 0 0 1 0
0 0 1 0 0
0 0 0 0 0

Difference Graph (g1-g2)::
Adjacency Matrix:
0 0 0 0 0
0 0 1 1 0
0 1 0 0 0
0 1 0 0 1
0 0 0 1 0

Difference Graph (g2-g1)::
Adjacency Matrix:
0 0 0 0 0
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0
0 1 1 0 0

Symmetric Difference Graph (Ring Sum) ::
Adjacency Matrix:
0 0 0 0 0
0 0 1 1 1
0 1 0 0 1
0 1 0 0 1
0 1 1 1 0
```

AIM 1 using Adjacency List::

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

void print(vector<vector<int>> adj) {

    cout << "Adjacency List: " << endl;
    for(int i=1;i<adj.size();i++) {
        cout << i << ": ";
        for(int j=0;j<adj[i].size();j++) {
            cout<<adj[i][j]<<" ";
        }
        cout << endl;
    }
    cout << endl;
}

vector<vector<int>> createGraph(int n, int e) {
    vector<vector<int>> adj(n+1);

    for(int i=0;i<e;i++) {
        int u,v;
        cin>>u>>v;

        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    return adj;
}

// union of two graphs
vector<vector<int>> unionGraph(vector<vector<int>> adj1,vector<vector<int>>
adj2) {

    int s1 = adj1.size();
    int s2 = adj2.size();
    int r;

    vector<vector<int>> uni;
    if(s1 > s2) {
```

```
        uni = adj1;
        r = s2;
    }
    else {
        uni = adj2;
        r = s1;
    }

    for(int i=1;i<r;i++) {
        for(int j=0;j<adj2[i].size();j++) {
            if(find(uni[i].begin(),uni[i].end(),adj2[i][j]) == uni[i].end())
                uni[i].push_back(adj2[i][j]);
        }
    }

    return uni;
}

// intersection of two graphs
vector<vector<int>> interSection(vector<vector<int>> adj1,vector<vector<int>>
adj2) {

    int s1 = adj1.size();
    int s2 = adj2.size();
    int r = max(s1,s2);



    vector<vector<int>> inter(r);

    for(int i=1;i<r;i++) {
        for(int j=0;j<adj2[i].size();j++) {
            if(find(adj1[i].begin(),adj1[i].end(),adj2[i][j]) !=
adj1[i].end())
                inter[i].push_back(adj2[i][j]);
        }
    }

    return inter;
}
```

```
// difference of two graphs
vector<vector<int>> diffGraph(vector<vector<int>> adj1,vector<vector<int>>
adj2) {

        int s1 = adj1.size();
        int s2 = adj2.size();
        int r = max(s1,s2);

        vector<vector<int>> diff(r);

        for(int i=1;i<r;i++) {
            for(int j=0;j<adj2[i].size();j++) {
                if(find(adj1[i].begin(),adj1[i].end(),adj2[i][j]) ==
adj1[i].end())
                    diff[i].push_back(adj2[i][j]);
            }
        }

        return diff;
}

// symmetric difference of two graphs
vector<vector<int>> symDiff(vector<vector<int>> adj1,vector<vector<int>> adj2)
{

    int s1 = adj1.size();
    int s2 = adj2.size();
    int r = max(s1,s2);

    vector<vector<int>> sym(r);

    for(int i=1;i<r;i++) {
        for(int j=0;j<adj2[i].size();j++) {
            if(find(adj1[i].begin(),adj1[i].end(),adj2[i][j]) ==
adj1[i].end())
                sym[i].push_back(adj2[i][j]);
        }
    }

    for(int i=1;i<r;i++) {
        for(int j=0;j<adj1[i].size();j++) {
```

```
            if(find(adj2[i].begin(),adj2[i].end(),adj1[i][j]) ==
adj2[i].end())
                sym[i].push_back(adj1[i][j]);
        }
    }
    return sym;
}

int main() {
    int n,e;
    cin>>n>>e;
    vector<vector<int>> g1 = createGraph(n,e);
    print(g1);

    int n1,e1;
    cin>>n1>>e1;
    vector<vector<int>> g2 = createGraph(n1,e1);
    print(g2);

    vector<vector<int>> uni = unionGraph(g1,g2);
    cout << "Union graph: " << endl;
    print(uni);

    vector<vector<int>> inter = interSection(g1,g2);
    cout << "Intersection graph: " << endl;
    print(inter);

    vector<vector<int>> diff = diffGraph(g1,g2);
    cout << "Difference graph(g2-g1): " << endl;
    print(diff);

    vector<vector<int>> diff2 = diffGraph(g2,g1);
    cout << "Difference graph(g1-g2): " << endl;
    print(diff2);

    vector<vector<int>> sym = symDiff(g1,g2);
    cout << "Symmetric difference (Ring Sum) graph: " << endl;
    print(sym);

    return 0;
}
```

Input:

5 6
1 2
1 3
2 3
2 4
3 4
4 5

5 5
1 2
4 3
5 3
2 5
3 1

Output:

```
5 6
1 2
1 3
2 3
2 4
3 4
4 5
Adjacency List:
1: 2 3
2: 1 3 4
3: 1 2 4
4: 2 3 5
5: 4

5 5
1 2
4 3
5 3
2 5
3 1
Adjacency List:
1: 2 3
2: 1 5
3: 4 5 1
4: 3
5: 3 2

Union graph:
Adjacency List:
1: 2 3
2: 1 5
3: 4 5 1
4: 3
5: 3 2
```

```
Intersection graph:
Adjacency List:
1: 2 3
2: 1
3: 4 1
4: 3
5:

Difference graph(g2-g1):
Adjacency List:
1:
2: 5
3: 5
4:
5: 3 2

Difference graph(g1-g2):
Adjacency List:
1:
2: 3 4
3: 2
4: 2 5
5: 4

Symmetric difference (Ring Sum) graph:
Adjacency List:
1:
2: 5 3 4
3: 5 2
4: 2 5
5: 3 2 4
```

AIM 2 (Path Generation)::

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

string direction = "DLRU";
int dr[4] = {1, 0, 0, -1};
int dc[4] = {0, -1, 1, 0};

bool isValid(int r, int c, int n, vector<vector<bool>>& maze) {
    return r >= 0 && c >= 0 && r < n && c < n && maze[r][c];
}

void printMaze(vector<vector<bool>>& maze) {
    for (int i = 0; i < maze.size(); i++) {
        for (int j = 0; j < maze[i].size(); j++)
            cout << maze[i][j] << " ";
        cout << endl;
    }
}

void findPath(pair<int, int> s, pair<int, int> end, vector<vector<bool>>&
maze, int n, vector<string>& ans, string& currentPath) {
    if (s.first == end.first && s.second == end.second) {
        ans.push_back(currentPath);
        return;
    }

    int r = s.first;
    int c = s.second;

    maze[r][c] = 0;

    for (int i = 0; i < 4; i++) {
        int nextr = r + dr[i];
        int nextc = c + dc[i];
        if (isValid(nextr, nextc, n, maze)) {
            currentPath += direction[i];
            findPath({nextr, nextc}, end, maze, n, ans, currentPath);
            currentPath.pop_back();
```

```
        }
    }
    maze[r][c] = 1;
}

int main() {
    vector<vector<bool>> maze = {
        {1, 0, 0, 1, 1},
        {1, 1, 1, 0, 0},
        {1, 0, 1, 1, 1},
        {1, 0, 1, 0, 1},
        {1, 1, 1, 1, 1}
    };

    int n = maze.size();
    cout << "Maze:: " << endl;
    printMaze(maze);

    int sX, sY, eX, eY;
    cout << "Enter start point: ";
    cin >> sX >> sY;
    cout << "Enter end point: ";
    cin >> eX >> eY;

    vector<string> result;
    string currentPath = "";

    findPath({sX, sY}, {eX, eY}, maze, n, result, currentPath);

    if (result.size() == 0)
        cout << "No path found" << endl;
    else{
        cout << "Path found: " << endl;
        for (int i = 0; i < result.size(); i++)
            cout << result[i] << " ";
    }

    return 0;
}
```

Input / Output:

Example - 1 :

```
Maze::
1 0 0 1 1
1 1 1 0 0
1 0 1 1 1
1 0 1 0 1
1 1 1 1 1
Enter start point: 2 3
Enter end point: 3 4
Path found:
LDDRRU LULLDDDRRRRU RD
```

Example - 2 :

```
Maze::
1 0 0 1 1
1 1 1 0 0
1 0 1 1 1
1 0 1 0 1
1 1 1 1 1
Enter start point: 0 0
Enter end point: 0 3
No path found
```