****

**LAB EXERCISE 8**

Name: Jayannthan P T

Dept: CSE ‘A’

Roll No.: 205001049



1. To Implement Warshall’s Algorithm Transitive Closure using DP
   1. **Code:**

#include <bits/stdc++.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define INF 9999;

using **namespace** std;

typedef **struct** Graph \*graph;

typedef **struct** Graph

{

**int** nv;

**int** \*\*am;

} Graph;

graph creategraph(**int** v)

{

    graph g = (graph)malloc(sizeof(Graph));

    g->nv = v;

    g->am = (**int** \*\*)malloc(v \* sizeof(**int** \*));

    for (**int** i = 0; i < v; i++)

    {

        g->am[i] = (**int** \*)malloc(v \* sizeof(**int**));

    }

    for (**int** i = 0; i < v; i++)

    {

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

        {

            g->am[i][j] = 0;

        }

    }

    return g;

}

graph fillmatrix(graph g, **int** i, **int** j)

{

    if (i < g->nv && j < g->nv)

    {

        g->am[i][j] = 1;

*// g->am[j][i] = w;*

    }

    return g;

}

graph getgraph(graph g)

{

**char** v1, v2;

**int** width;

    printf("\nEdge ::\n Vertice 1 :: ");

    cin >> v1;

    printf(" Vertice 2 :: ");

    cin >> v2;

    while (v1 != '0' && v2 != '0')

    {

**int** vv1 = v1 - 'A';

**int** vv2 = v2 - 'A';

        g = fillmatrix(g, vv1, vv2);

        printf("\nEdge ::\n Vertice 1 :: ");

       cin >> v1;

        printf(" Vertice 2 :: ");

        cin >> v2;

    }

    return g;

}

**void** warshalls(graph g)

{

**int** D[g->nv][g->nv];

    for (**int** i = 0; i < g->nv; i++)

    {

        for (**int** j = 0; j < g->nv; j++)

        {

            D[i][j] = g->am[i][j];

        }

    }

    for (**int** k = 0; k < g->nv; k++)

    {

        for (**int** i = 0; i < g->nv; i++)

        {

            for (**int** j = 0; j < g->nv; j++)

            {

                if (D[i][j] == 1 || (D[i][k] && D[k][j]))

                {

                    D[i][j] = 1;

                }

            }

        }

    }

    cout << "\nWarshal;s transitive closure ::\n";

    for (**int** i = 0; i < g->nv; i++)

    {

        for (**int** j = 0; j < g->nv; j++)

        {

            if (D[i][j] == 1)

                cout << **char**(j + 'A') << ", ";

        }

        cout << "--- is reachable from " << **char**(i + 'A') << endl;

    }

    cout << "\n";

    for (**int** i = 0; i < g->nv; i++)

    {

        for (**int** j = 0; j < g->nv; j++)

        {

            cout << D[i][j] << " ";

        }

        cout << endl;

    }

}

**int** main()

{

    graph g;

**int** n, src;

**char** ch;

    printf("\nEnter no. of vertices:");

    cin >> n;

    g = (graph)malloc(sizeof(Graph));

    g = NULL;

    g = creategraph(n);

    printf("\nEnter the Edges (Enter '0 0 0' to exit) ::\n");

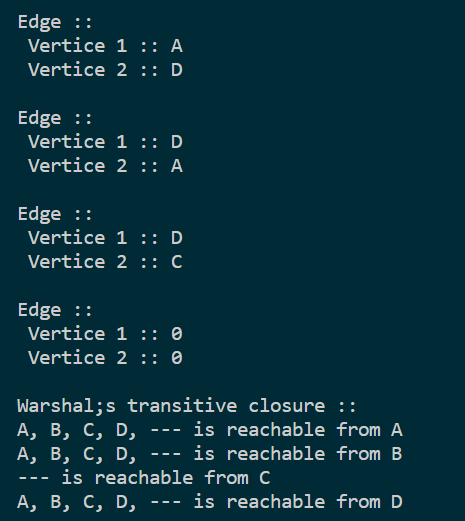
    g = getgraph(g);

    warshalls(g);

    return 0;

}

* 1. **Output:**

****



1. To Implement Floyd’s Algorithm for all pair shortest path using DP
   1. **Code:**

#include <bits/stdc++.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define INF 9999;

using **namespace** std;

typedef **struct** Graph \*graph;

typedef **struct** Graph

{

**int** nv;

**int** \*\*am;

} Graph;

graph creategraph(**int** v)

{

    graph g = (graph)malloc(sizeof(Graph));

    g->nv = v;

    g->am = (**int** \*\*)malloc(v \* sizeof(**int** \*));

    for (**int** i = 0; i < v; i++)

    {

        g->am[i] = (**int** \*)malloc(v \* sizeof(**int**));

    }

    for (**int** i = 0; i < v; i++)

    {

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

        {

            if (i == j)

                g->am[i][j] = 0;

            else

                g->am[i][j] = INF;

        }

    }

    return g;

}

graph fillmatrix(graph g, **int** i, **int** j, **int** w)

{

    if (i < g->nv && j < g->nv)

    {

        g->am[i][j] = w;

*// g->am[j][i] = w;*

    }

    return g;

}

graph getgraph(graph g)

{

**char** v1, v2;

**int** width;

    printf("\nEdge ::\n Vertice 1 :: ");

    cin >> v1;

    printf(" Vertice 2 :: ");

    cin >> v2;

    printf(" Weight of edge :: ");

    cin >> width;

    while (v1 != '0' && v2 != '0')

    {

**int** vv1 = v1 - 'A';

**int** vv2 = v2 - 'A';

        g = fillmatrix(g, vv1, vv2, width);

        printf("\nEdge ::\n Vertice 1 :: ");

        cin >> v1;

        printf(" Vertice 2 :: ");

        cin >> v2;

        printf(" Weight of edge :: ");

        cin >> width;

    }

    return g;

}

**void** floyd(graph g)

{

**int** D[g->nv][g->nv];

    for (**int** i = 0; i < g->nv; i++)

    {

        for (**int** j = 0; j < g->nv; j++)

        {

            D[i][j] = g->am[i][j];

        }

    }

    for (**int** k = 0; k < g->nv; k++)

    {

        for (**int** i = 0; i < g->nv; i++)

        {

            for (**int** j = 0; j < g->nv; j++)

            {

                D[i][j] = min(D[i][j], D[i][k] + D[k][j]);

            }

        }

    }

    cout << "\nFloyd's shortest path ::\n ";

    for (**int** i = 0; i < g->nv; i++)

    {

        for (**int** j = 0; j < g->nv; j++)

        {

            if (i == j)

                continue;

            cout << **char**(i + 'A') << " --> " << **char**(j + 'A') << " ";

            cout << D[i][j];

            cout << "\n ";

        }

        cout << "\n ";

    }

}

**int** main()

{

    graph g;

**int** n, src;

**char** ch;

    printf("\nEnter no. of vertices:");

    cin >> n;

    g = (graph)malloc(sizeof(Graph));

    g = NULL;

    g = creategraph(n);

    printf("\nEnter the Edges (Enter '0 0 0' to exit) ::\n");

    g = getgraph(g);

    floyd(g);

    return 0;

}

* 1. **Output:**

