PRACTICAL – 6

AIMAN SIDDIQUA **2K18/MC/008**

AIM: To write a program to find the shortest path between every pair of vertices in a graph using Floyd-Warshall's Algorithm.

CODE:

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

#define N 4
#define INF 100000

void floydWarshall(int g[][N])
{
    int dist[N][N], i, j, k;

    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            dist[i][j] = g[i][j];

    for (k = 0; k < N; k++) {
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                if (dist[i][j] > (dist[i][k] + dist[k][j])
                    && (dist[k][j] != INF
                        && dist[i][k] != INF))
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (dist[i][j] == INF)
                cout << "INF"
                     << " ";
            else
                cout << dist[i][j] << " ";
        }
        cout << endl;
    }
}
```

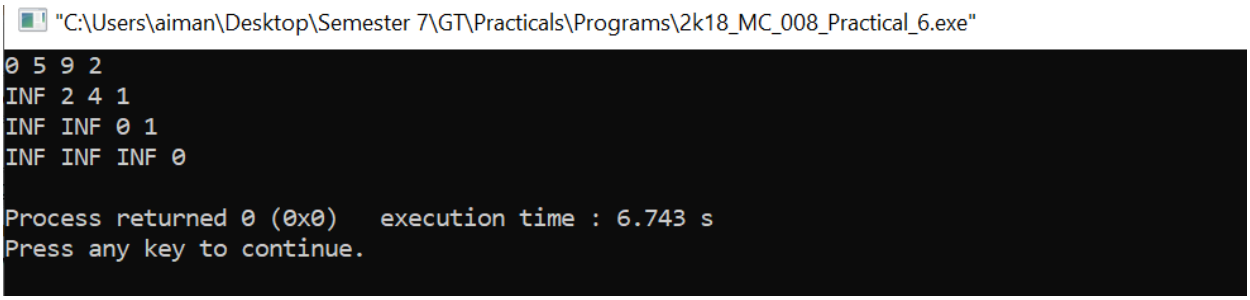
```
int main(){

    int g[4][4] = {{0 ,5,INF,2},
                   {INF,2,4,1},
                   {INF,INF,0,1},
                   {INF,INF,INF,0}};

    floydWarshall(g);

    return 0;
}
```

OUTPUT:



PRACTICAL – 7

AIMAN SIDDIQUA **2K18/MC/008**

AIM: To write a program to find the shortest path between two vertices in a graph using Bellaman Ford's Algorithm.

CODE:

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

struct Edge {
    int src, dest, weight;
};

struct Graph {
    int V, E;
    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[E];
    return graph;
}

void BellmanFord(struct Graph* graph, int src)
{
    int V = graph->V;
    int E = graph->E;
    int dist[V];

    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;
    dist[src] = 0;

    for (int i = 1; i <= V - 1; i++) {
        for (int j = 0; j < E; j++) {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }
```

```
    for (int i = 0; i < E; i++) {
        int u = graph->edge[i].src;
        int v = graph->edge[i].dest;
        int weight = graph->edge[i].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
            printf("Graph contains negative weight cycle");
            return;
        }
    }

    printf("Vertex Distance from Source\n");
    for (int i = 0; i < V; ++i)
        printf("%d \t\t %d\n", i, dist[i]);

}

int main()
{
    int V = 5;
    int E = 8;
    struct Graph* graph = createGraph(V, E);

    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = -1;

    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 4;

    graph->edge[2].src = 1;
    graph->edge[2].dest = 2;
    graph->edge[2].weight = 3;

    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 2;

    graph->edge[4].src = 1;
    graph->edge[4].dest = 4;
    graph->edge[4].weight = 2;

    graph->edge[5].src = 3;
    graph->edge[5].dest = 2;
    graph->edge[5].weight = 5;

    graph->edge[6].src = 3;
    graph->edge[6].dest = 1;
    graph->edge[6].weight = 1;

    graph->edge[7].src = 4;
    graph->edge[7].dest = 3;
```

```
    graph->edge[7].weight = -3;

    BellmanFord(graph, 0);

    return 0;
}
```

OUTPUT: