WEEK- 7 .

PRIYANSHI

2016927

E – 26

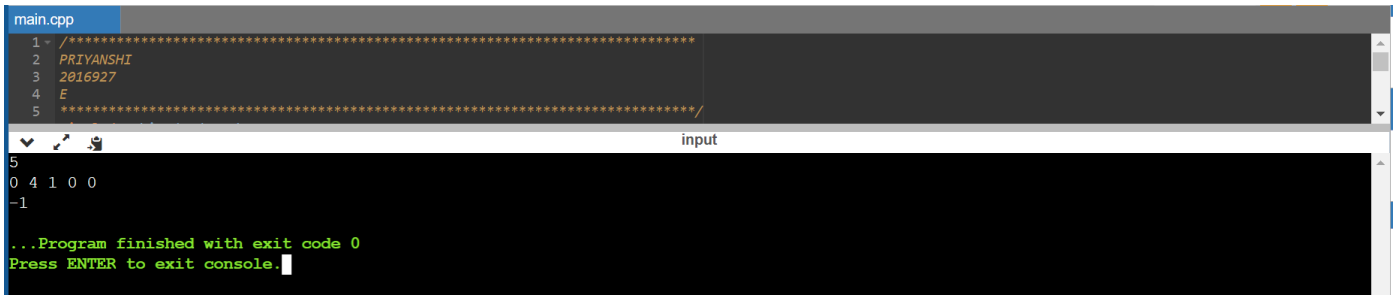
Q1.

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


int bellman_ford(int **graph, int source, int destination, int vertices) {
    int dist[vertices];
    for (int i = 0; i < vertices; i++) {
        dist[i] = INT_MAX;
    }
    dist[source] = 0;
    for (int i = 0; i < vertices - 1; i++) {
        for (int j = 0; j < vertices; j++) {
            for (int k = 0; k < vertices; k++) {
                if (dist[j] != INT_MAX && graph[j][k] != 0 && dist[j] + graph[j][k] < dist[k]) {
                    dist[k] = dist[j] + graph[j][k];
                }
            }
        }
    }
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
```

```
            if (dist[j] != INT_MAX && graph[j][i] != 0 && dist[j] + graph[j][i] < dist[i]) {
                return -1;
            }
        }
    }
    return dist[destination];
}
int main()
{
    int vertices, edges;
    cin >> vertices >> edges;
    int **graph = new int *[vertices];
    for (int i = 0; i < vertices; i++) {
        graph[i] = new int[vertices];
        for (int j = 0; j < vertices; j++) {
            graph[i][j] = 0;
        }
    }
    for (int i = 0; i < edges; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u][v] = w;
    }
    int source, destination;
    cin >> source >> destination;
```

```
    int ans = bellman_ford(graph, source, destination, vertices);
    if (ans == INT_MAX) {
        cout << "-1";
    } else {
        cout << ans;
    }
    return 0;
}
```

OUTPUT



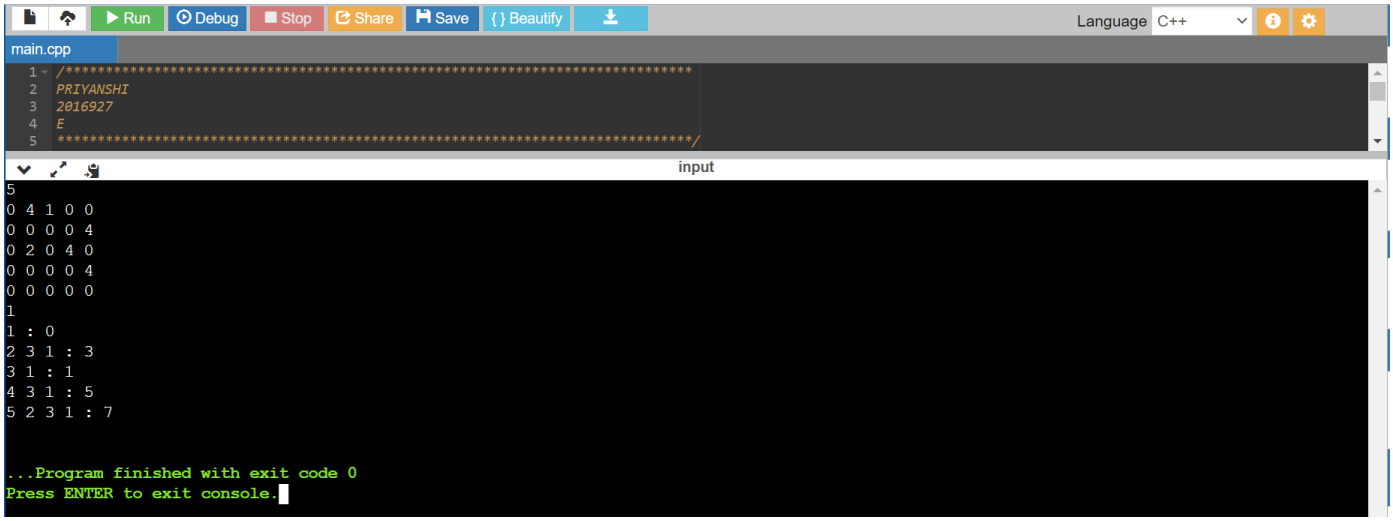
Q2.

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

void calulate(vector<int> &pa, int i) {
    cout << i + 1 << " ";
    if (pa[i] >= 0)
        calulate(pa, pa[i]);
}

void find_path(int **graph, int m, int sour) {
    vector<int> dis(m, INT_MAX), pa(m, -1);
```

```
    dis[sour] = 0;
    for (int ki = 0; ki < m - 1; ki++) {
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < m; j++) {
                if (graph[i][j] != 0) {
                    if (dis[j] > dis[i] + graph[i][j]) {
                        dis[j] = dis[i] + graph[i][j];
                        pa[j] = i;
                    }
                }
            }
        }
    }
    for (int i = 0; i < m; i++) {
        calulate(pa, i);
        cout << ": " << dis[i] << endl;
    }
}

int main() {
    int m, source, ed;
    cin >> m;
    int **graph = (int **)malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++)
        graph[i] = (int *)malloc(m * sizeof(int));
```

```
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            cin >> graph[i][j];
        }
    }
    cin >> source;
    find_path(graph, m, source - 1);
}
```

OUTPUT



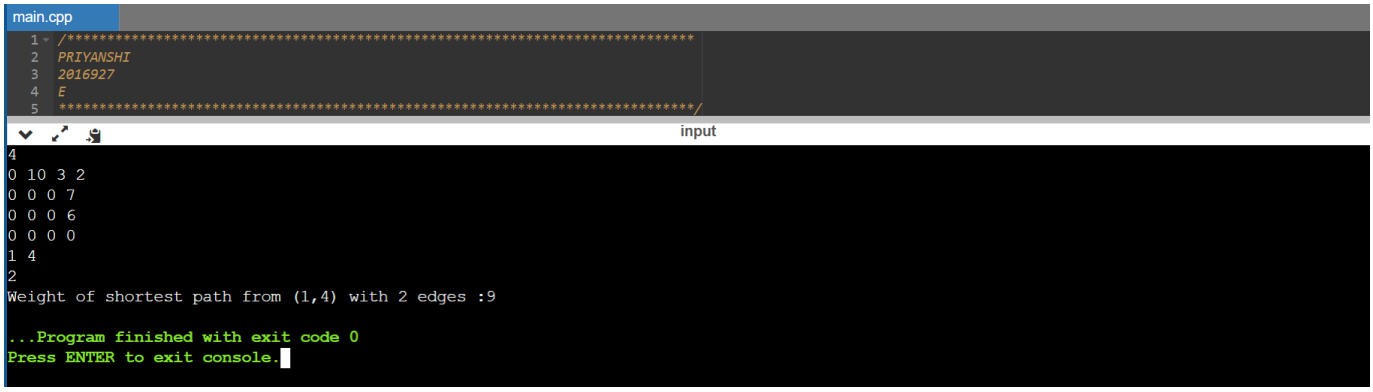
Q3.

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

int shortest_weigt(int **graph, int ver, int u, int v, int k) {
    if (k <= 0)
        return INT_MAX;
    if (k == 0 && u == v)
        return 0;
```

```
    if (k == 1 && graph[u][v] != INT_MAX)
        return graph[u][v];
    int res = INT_MAX;
    for (int i = 0; i < ver; i++) {
        if (graph[u][i] != 0 && u != i && v != i) {
            int recu = shortest_weigt(graph, ver, i, v, k - 1);
            if (recu != INT_MAX)
                res = min(res, graph[u][i] + recu);
        }
    }
    return res;
}

int main() {
    int ver, u, v, k, ans;
    cin >> ver;
    int **graph = (int **)malloc(ver * sizeof(int *));
    for (int i = 0; i < ver; i++)
        graph[i] = (int *)malloc(sizeof(int) * ver);
    for (int i = 0; i < ver; i++)
        for (int j = 0; j < ver; j++)
            cin >> graph[i][j];
    cin >> u >> v >> k;
    ans = shortest_weigt(graph, ver, u - 1, v - 1, k);
    cout << "Weight of shortest path from (" << u<< "," << v << ") with " << k << " edges :" <<
ans;
```

}

OUTPUT


main.cpp
/***
PRIYANSHI
2016927
E
***/
input
4
0 10 3 2
0 0 0 7
0 0 0 6
0 0 0 0
1 4
2
Weight of shortest path from (1,4) with 2 edges :9
...Program finished with exit code 0
Press ENTER to exit console.