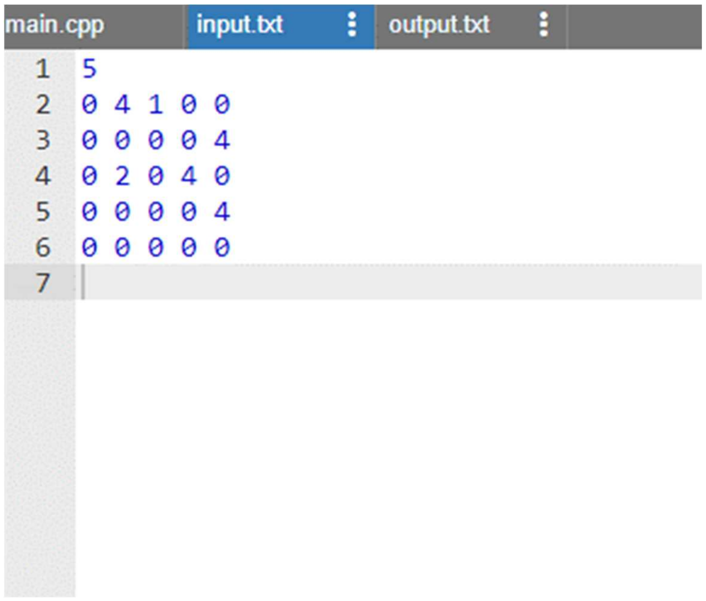
DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

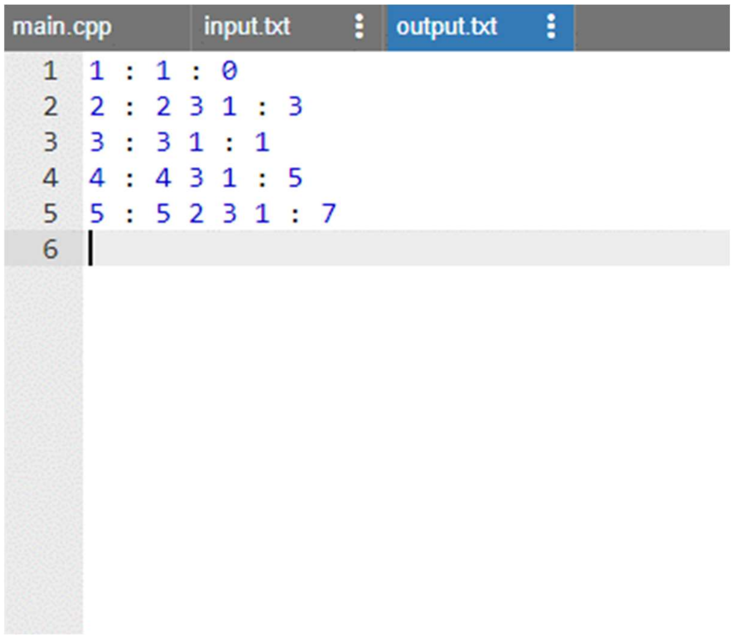
WEEK 7

1. After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay's house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends' location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend's location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra's algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay's results? (Hint: Print shortest path and distance from friends' location to Akshay's house)

```
#include <iostream>
#include <vector>
#include <fstream>
#include<queue>
#include<climits>
using namespace std;
void printPath(vector<int>& parent, int v, int src) {
    if (v == src) {
        cout << v + 1;
         return;
    }
    cout << v + 1 << " ";
    printPath(parent, parent[v], src);
}

void printSolution(vector<int>& dist, vector<int>& parent, int src,ofstream&fout) {
    for (int i = 0; i < dist.size(); ++i) {
        fout << i + 1 << " : ";
        if (dist[i] == INT_MAX)
            cout << "No path";
        else {
            printPath(parent, i, src);
        }
        fout << " : " << dist[i];
        fout << endl;
    }
}

void dijkstra(vector<vector<int>>& graph, int src,ofstream &fout) {
    int V = graph.size();
    vector<int> dist(V, INT_MAX);
    vector<int> parent(V, -1);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    dist[src] = 0;
    pq.push({0, src});
    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        for (int v = 0; v < V; ++v) {
            if (graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
                parent[v] = u;
                pq.push({dist[v], v});
            }
        }
    }
    printSolution(dist, parent, src,fout);
}



int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int V;fin>>V;
    vector<vector<int>> graph(V, vector<int>(V));
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            int weight;
            fin >> weight;
            graph[i][j] = weight;
        }
    }
    int src;fin>>src;
    src--;
    dijkstra(graph, src,fout);

    fin.close();
    fout.close();

    return 0;
}
```



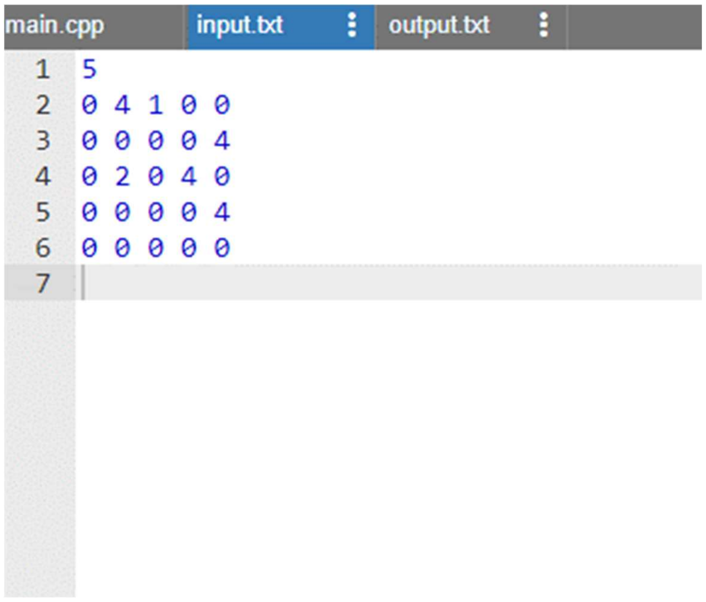
```
5
0 4 1 0 0
0 0 0 0 4
0 2 0 4 0
0 0 0 0 4
0 0 0 0 0
```



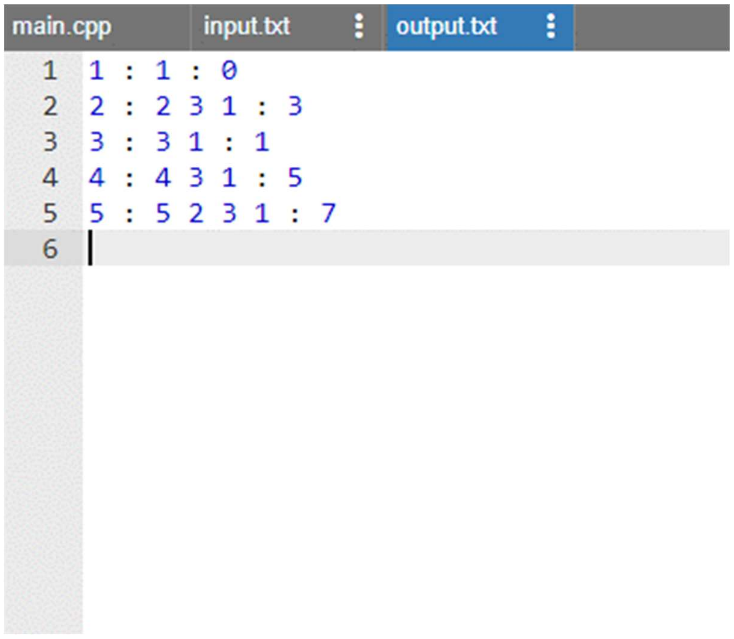
```
1 : 1 : 0
2 : 2 3 1 : 3
3 : 3 1 : 1
4 : 4 3 1 : 5
5 : 5 2 3 1 : 7
```

2. Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.

```
#include <iostream>
#include <vector>
#include <fstream>
#include<queue>
#include<climits>
using namespace std;
void printPath(vector<int>& parent, int v, int src) {
    if (v == src) {
        cout << v + 1;
         return;
    }
    cout << v + 1 << " ";
    printPath(parent, parent[v], src);
}

void printSolution(vector<int>& dist, vector<int>& parent, int src,ofstream&fout) {
    for (int i = 0; i < dist.size(); ++i) {
        fout << i + 1 << " : ";
        if (dist[i] == INT_MAX)
            cout << "No path";
        else {
            printPath(parent, i, src);
        }
        fout << " : " << dist[i];
        fout << endl;
    }
}

void bellmanFord(vector<vector<int>>& graph, int src) {

    int V = graph.size();
    vector<int> dist(V, INF);
    vector<int> parent(V, -1);
    dist[src] = 0;
    for (int i = 1; i < V; ++i) {
        for (int u = 0; u < V; ++u) {
            for (int v = 0; v < V; ++v) {
                if (graph[u][v] && dist[u] != INF && dist[u] + graph[u][v] < dist[v]) {
                        dist[v] = dist[u] + graph[u][v];
                        parent[v] = u;
                }
            }
        }
    }
    for (int u = 0; u < V; ++u) {
        for (int v = 0; v < V; ++v) {
            if (graph[u][v] && dist[u] != INF && dist[u] + graph[u][v] < dist[v]) {
                cout << "Graph contains negative cycle";
                return;
            }
        }
    }
    printSolution(dist, parent, src,fout);
}




int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int V;fin>>V;
    vector<vector<int>> graph(V, vector<int>(V));
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            int weight;
            fin >> weight;
            graph[i][j] = weight;
        }
    }
    int src;fin>>src;
    src--;
    dijkstra(graph, src,fout);

    fin.close();
    fout.close();

    return 0;
}
```



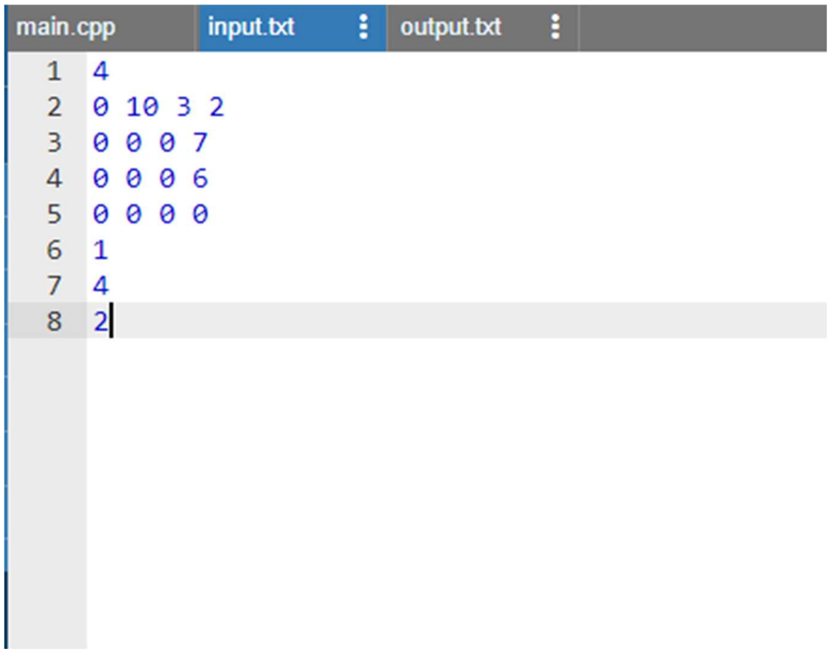
```
5
0 4 1 0 0
0 0 0 0 4
0 2 0 4 0
0 0 0 0 4
0 0 0 0 0
```



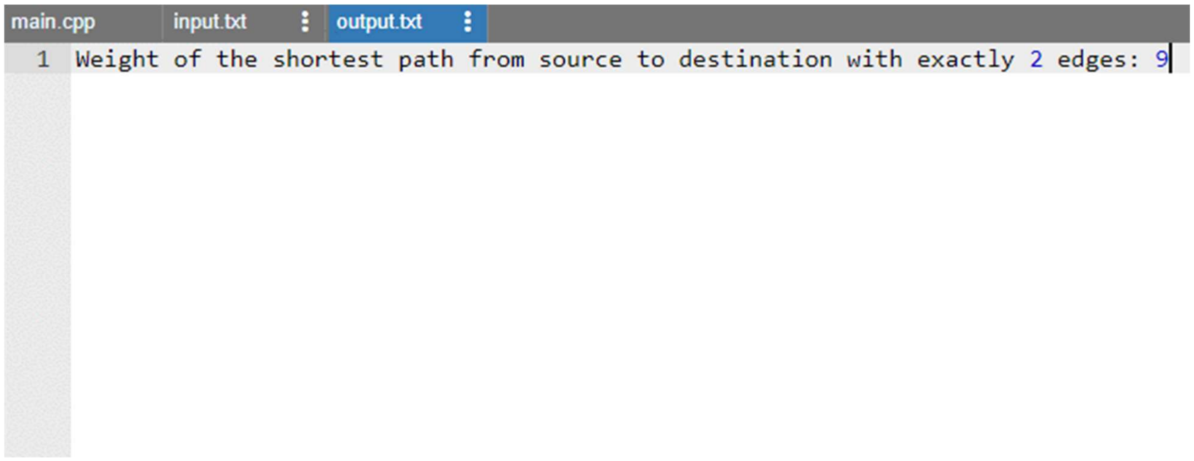
```
1 : 1 : 0
2 : 2 3 1 : 3
3 : 3 1 : 1
4 : 4 3 1 : 5
5 : 5 2 3 1 : 7
```

3. Given a directed graph with two vertices (source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.

```
#include <iostream>
#include <vector>
#include <fstream>
#include<queue>
#include<climits>
using namespace std;

int shortestPathWithKEdges(vector<vector<int>>& graph, int src, int dest, int k) {
    int V = graph.size();
    vector<vector<int>> dist(V, vector<int>(k + 1, INT_MAX));
    dist[src][0] = 0;
    for (int count = 0; count < k; ++count) {
        for (int u = 0; u < V; ++u) {
            for (int v = 0; v < V; ++v) {
                 if (graph[u][v] != 0 &&
                 dist[u][count] != INT_MAX &&
                 dist[u][count] + graph[u][v]
                 <dist[v][count + 1]) {
                                                                    dist[v][count + 1] = dist[u][count] +
graph[u][v];
                 }
            }
        }
    }
    for (int u = 0; u < V; ++u) {
        for (int v = 0; v < V; ++v) {
            if (graph[u][v] != 0 &&
            dist[u][k] != INT_MAX &&
            dist[u][k] + graph[u][v] < dist[v][k]) {
                return -1;
            }
        }
    }
    return dist[dest][k];
}






int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int V;fin>>V;
    vector<vector<int>> graph(V, vector<int>(V));
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            cin >> graph[i][j];
        }
    }
    int src, dest, k;
    fin>>src>>dest>>k;
    int weight = shortestPathWithKEdges(graph, src - 1, dest - 1, k);
    if (weight == -1) {
        fout << "Negative cycle detected,
        no shortest path with exactly" << k << " edges
        exists." << endl;
    }
    else if (weight == INT_MAX) {
        fout << "No path of length " << k << " is available." << endl;
    }
    else {
        fout << "Weight of the shortest path from source
        to destination with exactly " << k
        << " edges: " << weight << endl;
    }



    fin.close();
    fout.close();

    return 0;
}
```



```
4
0 10 3 2
0 0 0 7
0 0 0 6
0 0 0 0
1
4
2
```



```
Weight of the shortest path from source to destination with exactly 2 edges: 9
```