Green University of Bangladesh
Department of Computer Science and Engineering(CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2021), B.Sc. in CSE (Day)

LAB REPORT NO 01

Course Title: Algorithms Lab

Course Code: CSE 206 Section: DB

Student Details

	Name	ID
1.	MD.Mehedi Hassan Nayeem	213902045

Lab Date : _

Submission Date : _

Course Teacher's Name : **Md. Sultanul Islam Ovi**

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status

Marks: …… **Signature:.....**

Comments:..... **Date:.....**

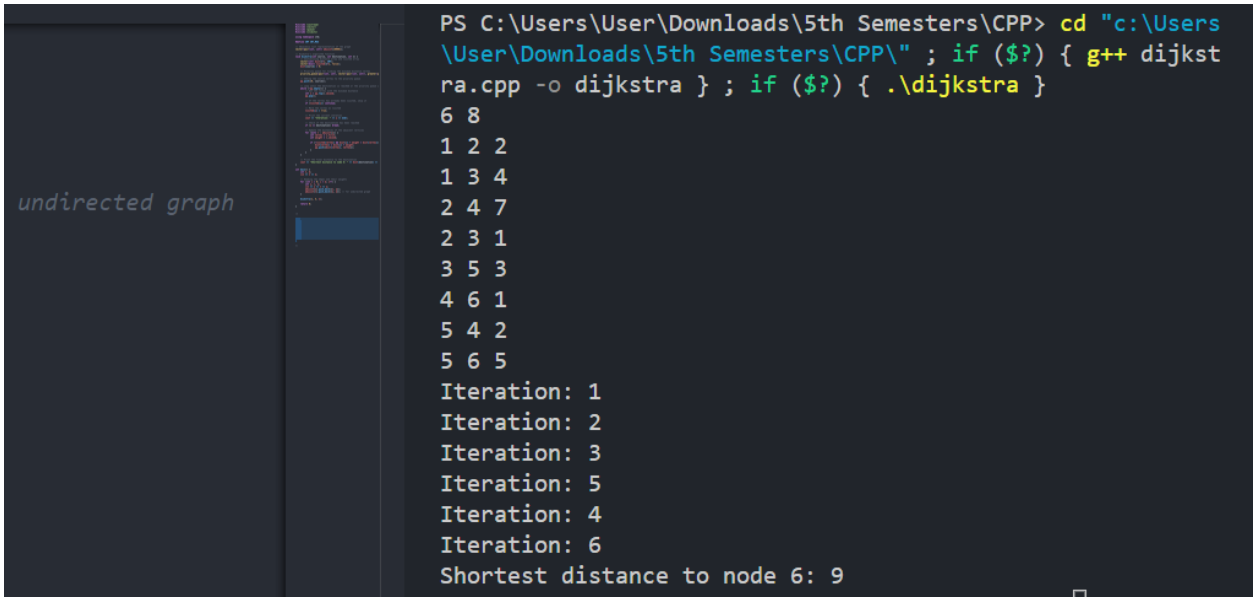
Problem 02 : Implement Dijkstra's algorithm for a single destination shortest path problem. Take node 6 as the destination. Also print each iteration.

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>

using namespace std;

#define INF INT_MAX

vector<pair<int, int>> adjList[100001];

void dijkstra(int source, int destination, int n) {

    vector<int> dist(n+1, INF);
    vector<bool> visited(n+1, false);
    dist[source] = 0;


    priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<pair<int, int>>> pq;

    pq.push({0, source});


    while (!pq.empty()) {

        int u = pq.top().second;
        pq.pop();


        if (visited[u]) continue;

        visited[u] = true;
```

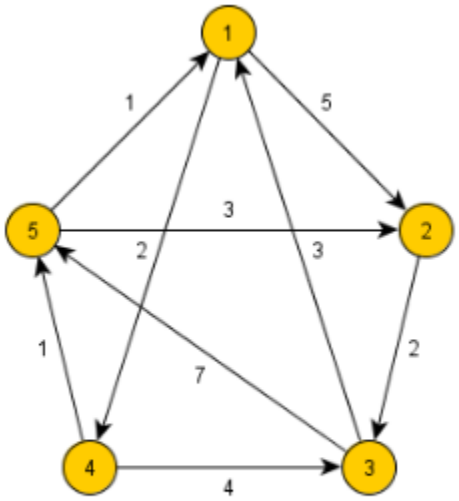
```
        cout << "Iteration: " << u << endl;


        if (u == destination) break;


        for (auto v : adjList[u]) {
            int vertex = v.first;
            int weight = v.second;

            if (!visited[vertex] && dist[u] + weight < dist[vertex])
{
                dist[vertex] = dist[u] + weight;
                pq.push({dist[vertex], vertex});
            }
        }
    }


    cout << "Shortest distance to node 6: " << dist[destination] <<
endl;
}

int main() {
    int n, m;
    cin >> n >> m;

    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adjList[u].push_back({v, w});
        adjList[v].push_back({u, w});     }

    dijkstra(1, 6, n);

    return 0;
}
```

```
/*

6 8
1 2 2
1 3 4
2 4 7
2 3 1
3 5 3
4 6 1
5 4 2
5 6 5
1

*/
```

Output :



Problem 03 : Implement Floyd Warshall on the following graph. Also print each iteration



Adjacency List :

1 -> (2,5),(4,2)
2-> (3,2)
3-> (1,3),(5,7)
4-> (3,4),(5,1)
5-> (1,1),(2,3)

Adjacency Matrix :

0 5 INF 2 INF
INF 0 2 INF INF
3 INF 0 INF 7
INF INF 4 0 1
1 3 INF INF 0

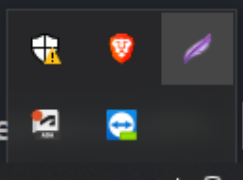
```cpp
#include<bits/stdc++.h>
using namespace std;

const int INF = 1e9;
const int MAXN = 505;

int dist[MAXN][MAXN];

void floyd_warshall(int n) {
    for(int k=1;k<=n;k++) {
        for(int i=1;i<=n;i++) {
            for(int j=1;j<=n;j++) {
                if(dist[i][k] != INF && dist[k][j] != INF &&
dist[i][j] > dist[i][k] + dist[k][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }

        cout << "Iteration #" << k << ":" << endl;
        for(int i=1;i<=n;i++) {
            for(int j=1;j<=n;j++) {
                if(dist[i][j] == INF) {
                    cout << "INF ";
                } else {
                    cout << dist[i][j] << " ";
                }
            }
            cout << endl;
        }
        cout << endl;
    }
}

int main() {
    int n, m;
```

```
    cin >> n >> m;

    for(int i=1;i<=n;i++) {
        for(int j=1;j<=n;j++) {
            dist[i][j] = INF;
        }
        dist[i][i] = 0;
    }

    for(int i=1;i<=m;i++) {
        int u, v, w;
        cin >> u >> v >> w;
        dist[u][v] = w;
    }

    floyd_warshall(n);

    cout << "The shortest path distances between each pair of
nodes are:" << endl;
    for(int i=1;i<=n;i++) {
        for(int j=1;j<=n;j++) {
            if(dist[i][j] == INF) {
                cout << "INF ";
            } else {
                cout << dist[i][j] << " ";
            }
        }
        cout << endl;
    }
}
```

```
PS C:\Users\User\Downloads\5th Semesters\CPP> cd "c:\Users
\User\Downloads\5th Semesters\CPP\" ; if ($?) { g++ floyed
_warshall.cpp -o floyed_warshall } ; if ($?) { .\floyed_wa
rshall }
5 9
1 2 5
1 4 2
2 3 2
3 1 3
3 5 7
4 3 4
4 5 1
5 1 1
5 2 3
Iteration #1:
0 5 INF 2 INF
INF 0 2 INF INF
3 8 0 5 7
INF INF 4 0 1
1 3 INF 3 0

Iteration #2:
0 5 7 2 INF
INF 0 2 INF INF
3 8 0 5 7
INF INF 4 0 1
1 3 5 3 0
```

```
 Iteration #3:
 0 5 7 2 14
 5 0 2 7 9
 3 8 0 5 7
 7 12 4 0 1
 1 3 5 3 0

 Iteration #4:
 0 5 6 2 3
 5 0 2 7 8
 3 8 0 5 6
 7 12 4 0 1
 1 3 5 3 0

 Iteration #5:
 0 5 6 2 3
 5 0 2 7 8
 3 8 0 5 6
 2 4 4 0 1
 1 3 5 3 0
```

```
The shortest path distances between each pair of nodes are
:
0 5 6 2 3
5 0 2 7 8
3 8 0 5 6
2 4 4 0 1
1 3 5 3 0
PS C:\Users\User\Downloads\5th Seme
```



Problem 01:

```
#include <bits/stdc++.h>
using namespace std;
const int inf = 1e9;

int main() {
    // Take input from the user
    int n, m, source;
    cin >> n >> m >> source;

    vector<vector<pair<int, int>>> graph(n + 1);
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u].push_back({v, w});
    }

    // Initialize the distance vector
    vector<int> dist(n + 1, inf);
    dist[source] = 0;

    // Run the algorithm
    for (int i = 1; i <= n - 1; i++) {
        for (int u = 1; u <= n; u++) {
            for (auto [v, w] : graph[u]) {
                if (dist[u] != inf && dist[v] > dist[u] +
w) {
                    dist[v] = dist[u] + w;
                }
            }
        }
    }
```

```
    // Check for negative cycles
    bool has_negative_cycle = false;
    for (int u = 1; u <= n; u++) {
        for (auto [v, w] : graph[u]) {
            if (dist[u] != inf && dist[v] > dist[u] + w) {
                has_negative_cycle = true;
                break;
            }
        }
        if (has_negative_cycle) break;
    }

    // Output the results
    if (has_negative_cycle) {
        cout << "Negative cycle detected!" << endl;
    } else {
        cout << "Shortest distances from node " << source
<< ": ";
        for (int i = 1; i <= n; i++) {
            if (dist[i] != inf) {
                cout << dist[i] << " ";
            } else {
                cout << "INF ";
            }
        }
        cout << endl;
    }

    return 0;
}
```