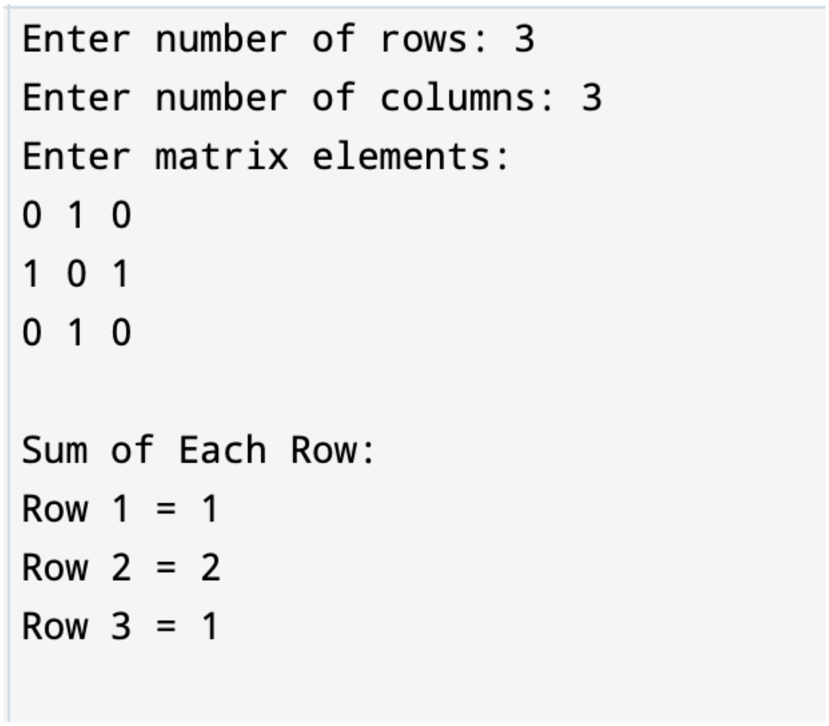
NAME – AADITA GARG
ROLL NO – 1024030461
SUB GROUP – 2C32

DATA STRUCTURES AND ALGORITHMS
ASSIGNMENT 9 – GRAPHS

(CLASS) Draw a graph using adjacency matrix and print sum of each row

```
#include <iostream>
using namespace std;

int main() {
   int r, c;
   cout << "Enter number of rows: ";
   cin >> r;
   cout << "Enter number of columns: ";
   cin >> c;

   int a[r][c];

   cout << "Enter matrix elements:\n";
   for (int i = 0; i < r; i++) {
      for (int j = 0; j < c; j++) {
         cin >> a[i][j];
      }
   }

   cout << "\nSum of Each Row:\n";
   for (int i = 0; i < r; i++) {
      int rowSum = 0;
      for (int j = 0; j < c; j++) {
         rowSum += a[i][j];
      }
```

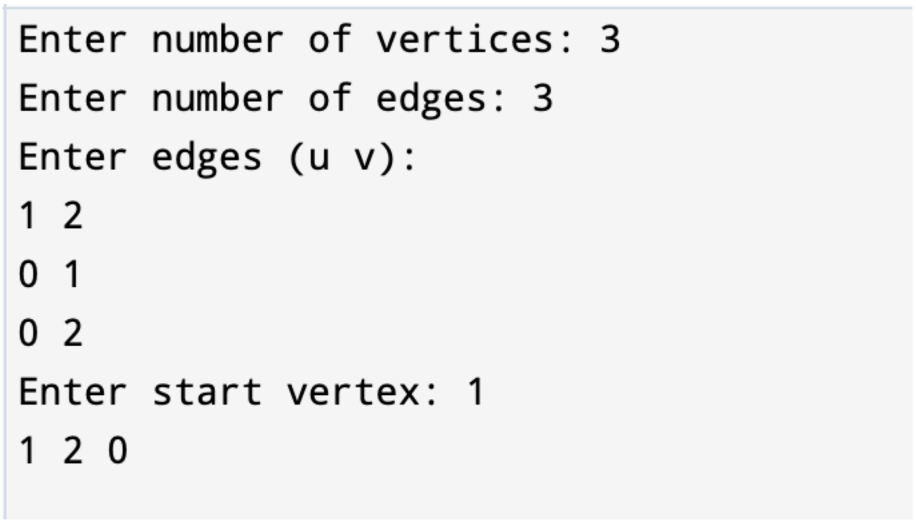
```
        cout << "Row " << i + 1 << " = " << rowSum << endl;
    }
    return 0;
}
```



```
Enter number of rows: 3
Enter number of columns: 3
Enter matrix elements:
0 1 0
1 0 1
0 1 0

Sum of Each Row:
Row 1 = 1
Row 2 = 2
Row 3 = 1
```

(1) Breadth First Search (BFS)

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

void BFS(int start, vector<vector<int>>& adj, int n) {
    vector<bool> visited(n, false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        cout << u << " ";

        for (int v : adj[u]) {
            if (!visited[v]) {
                visited[v] = true;
                q.push(v);
            }
        }
    }
}

int main() {
    int n, e;
    cout << "Enter number of vertices: ";
    cin >> n;

    vector<vector<int>> adj(n);
    cout << "Enter number of edges: ";
    cin >> e;

    cout << "Enter edges (u v):\n";
    for (int i = 0; i < e; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    int start;
    cout << "Enter start vertex: ";
    cin >> start;

    BFS(start, adj, n);
}
```

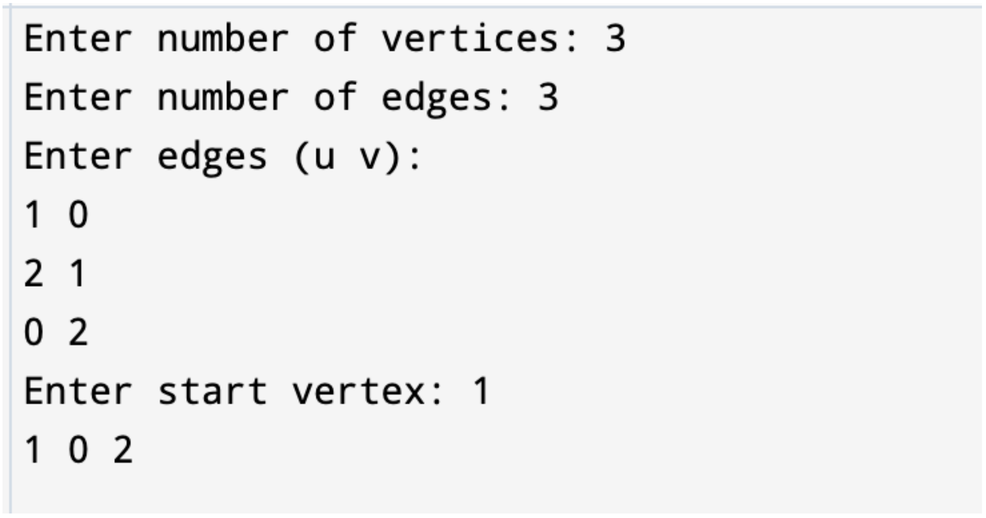


```
Enter number of vertices: 3
Enter number of edges: 3
Enter edges (u v):
1 2
0 1
0 2
Enter start vertex: 1
1 2 0
```

(2) Depth First Search (DFS)

```
#include <iostream>
#include <vector>
using namespace std;

void DFSUtil(int u, vector<vector<int>>& adj, vector<bool>& visited) {
   visited[u] = true;
   cout << u << " ";

   for (int v : adj[u]) {
      if (!visited[v])
         DFSUtil(v, adj, visited);
   }
}

int main() {
   int n, e;
   cout << "Enter number of vertices: ";
   cin >> n;

   vector<vector<int>> adj(n);
   cout << "Enter number of edges: ";
   cin >> e;
```

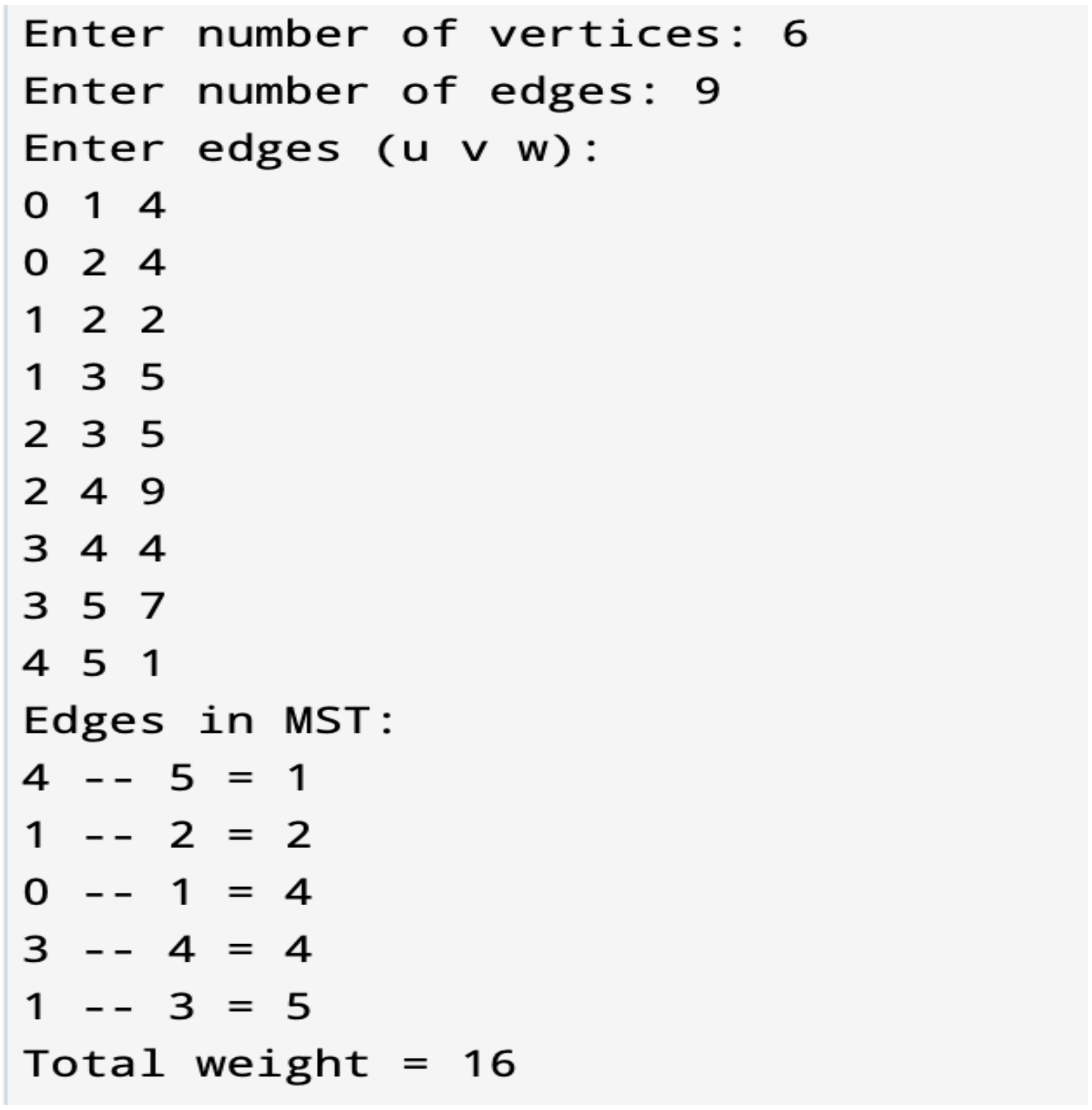
```
    cout << "Enter edges (u v):\n";
    for (int i = 0; i < e; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<bool> visited(n, false);

    int start;
    cout << "Enter start vertex: ";
    cin >> start;

    DFSUtil(start, adj, visited);
}
```



```
Enter number of vertices: 3
Enter number of edges: 3
Enter edges (u v):
1 0
2 1
0 2
Enter start vertex: 1
1 0 2
```

(3) Minimum Spanning Tree (Kruskal's Algorithm)

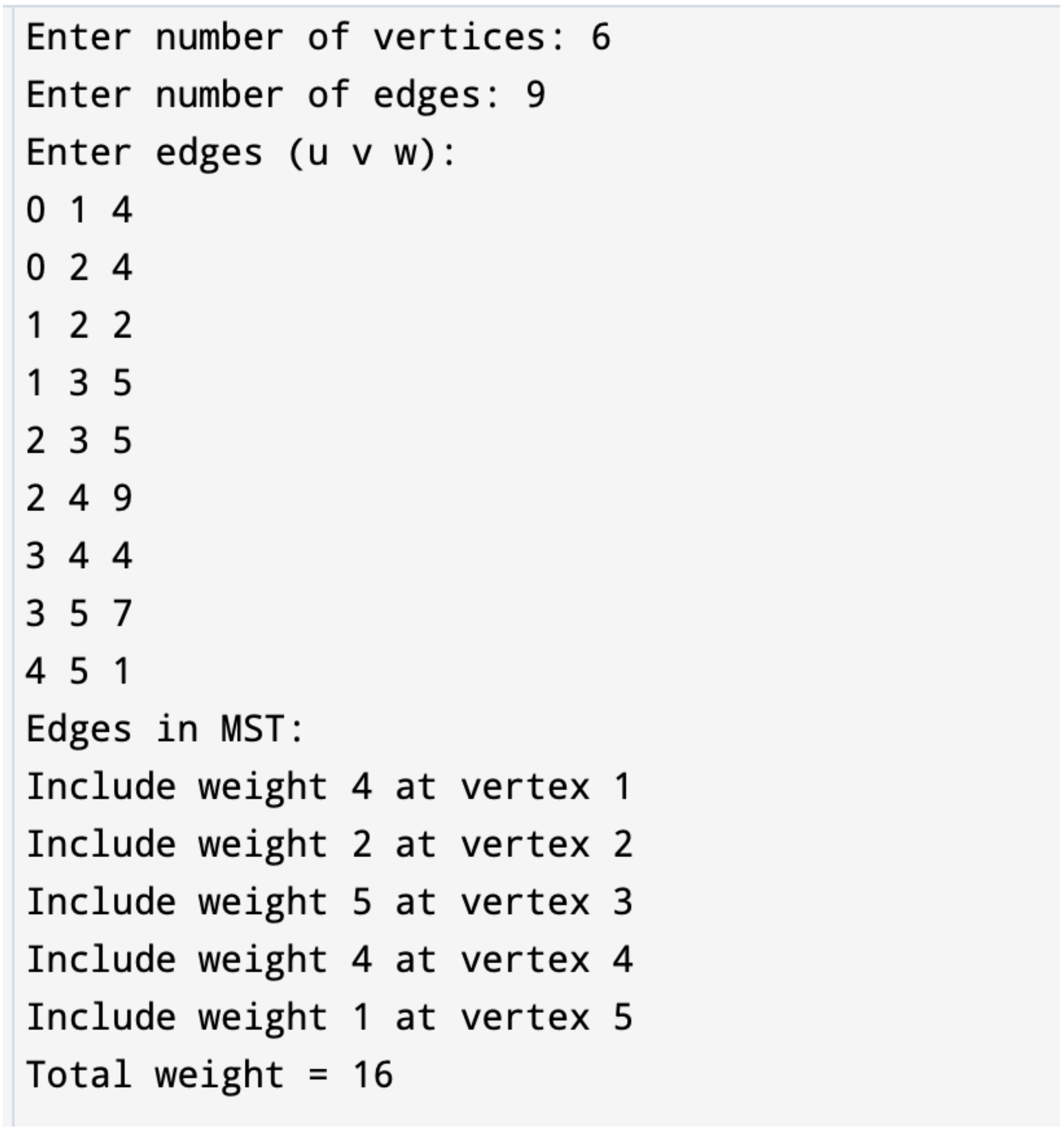
```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Edge {
    int u, v, w;
};

struct DSU {
    vector<int> parent, rank;

    DSU(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; i++)
            parent[i] = i;
    }

    int find(int x) {
        if (parent[x] != x)
            parent[x] = find(parent[x]);
        return parent[x];
    }

    void unite(int x, int y) {
        x = find(x);
        y = find(y);

        if (rank[x] < rank[y])
            parent[x] = y;
        else if (rank[x] > rank[y])
            parent[y] = x;
        else {
            parent[y] = x;
            rank[x]++;
        }
    }
};

int main() {
    int n, e;
    cout << "Enter number of vertices: ";
    cin >> n;

    cout << "Enter number of edges: ";
    cin >> e;

    vector<Edge> edges(e);
    cout << "Enter edges (u v w):\n";
    for (int i = 0; i < e; i++)
        cin >> edges[i].u >> edges[i].v >> edges[i].w;

    sort(edges.begin(), edges.end(), [](Edge a, Edge b) {
        return a.w < b.w;
    });

    DSU dsu(n);
    cout << "Edges in MST:\n";

    int mstWeight = 0;
    for (auto &edge : edges) {
        if (dsu.find(edge.u) != dsu.find(edge.v)) {
            cout << edge.u << " -- " << edge.v << " = " << edge.w <<
endl;
            mstWeight += edge.w;
            dsu.unite(edge.u, edge.v);
        }
    }

    cout << "Total weight = " << mstWeight;
}
```



```
Enter number of vertices: 6
Enter number of edges: 9
Enter edges (u v w):
0 1 4
0 2 4
1 2 2
1 3 5
2 3 5
2 4 9
3 4 4
3 5 7
4 5 1
Edges in MST:
4 -- 5 = 1
1 -- 2 = 2
0 -- 1 = 4
3 -- 4 = 4
1 -- 3 = 5
Total weight = 16
```

(4) Minimum Spanning Tree (Prim's Algorithm)

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

typedef pair<int, int> pii;  // (weight, vertex)

int main() {
    int n, e;
    cout << "Enter number of vertices: ";
    cin >> n;

    vector<vector<pii>> adj(n);
    cout << "Enter number of edges: ";
    cin >> e;

    cout << "Enter edges (u v w):\n";
    for (int i = 0; i < e; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({w, v});
        adj[v].push_back({w, u});
    }
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    vector<bool> inMST(n, false);

    pq.push({0, 0}); // (weight, start node)
    int mstWeight = 0;

    cout << "Edges in MST:\n";

    while (!pq.empty()) {
        auto [w, u] = pq.top();
        pq.pop();
        if (inMST[u]) continue;
        inMST[u] = true;
      mstWeight += w;
        if (w != 0)
            cout << "Include weight " << w << " at vertex " << u <<
endl;
        for (auto &p : adj[u]) {
            int weight = p.first;
            int v = p.second;

            if (!inMST[v])
                pq.push({weight, v}); } }
    cout << "Total weight = " << mstWeight; }
```



```
Enter number of vertices: 6
Enter number of edges: 9
Enter edges (u v w):
0 1 4
0 2 4
1 2 2
1 3 5
2 3 5
2 4 9
3 4 4
3 5 7
4 5 1
Edges in MST:
Include weight 4 at vertex 1
Include weight 2 at vertex 2
Include weight 5 at vertex 3
Include weight 4 at vertex 4
Include weight 1 at vertex 5
Total weight = 16
```

(5) Minimum Spanning Tree (Dijkstra’s Shortest Path Algorithm)

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

typedef pair<int, int> pii; // (distance, vertex)
```

```
int main() {
    int n, e;
    cout << "Enter number of vertices: ";
    cin >> n;

    vector<vector<pii>> adj(n);
    cout << "Enter number of edges: ";
    cin >> e;

    cout << "Enter edges (u v w):\n";
    for (int i = 0; i < e; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({w, v});
        adj[v].push_back({w, u}); // undirected graph
    }

    int start;
    cout << "Enter starting vertex: ";
    cin >> start;

    vector<int> dist(n, 1e9);
    priority_queue<pii, vector<pii>, greater<pii>> pq;

    dist[start] = 0;
    pq.push({0, start});

    while (!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();
        if (d != dist[u]) continue;

        for (auto &p : adj[u]) {
            int w = p.first;
            int v = p.second;

            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
    cout << "Shortest distances from " << start << ":\n";
    for (int i = 0; i < n; i++)
        cout << "to " << i << " = " << dist[i] << endl;
}
```

```
Shortest distances from 0:
to 0 = 0
to 1 = 3
to 2 = 1
to 3 = 4
to 4 = 7
to 5 = 9
```