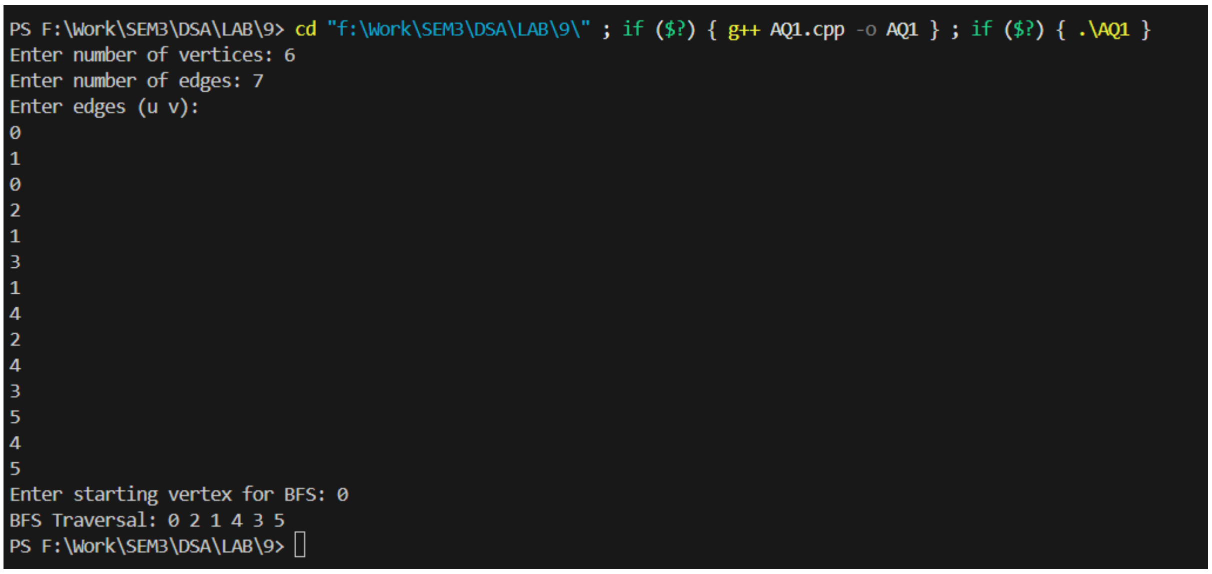
AQ1:

```c
#include <stdio.h>

#include <stdlib.h>

#define MAX 100

struct Node {
    int vertex;
    struct Node* next;
};

struct Node* adjList[MAX];
int visited[MAX];

int queue[MAX], front = -1, rear = -1;

void enqueue(int value) {
    if (rear == MAX - 1)
        return;
    if (front == -1) front = 0;
    queue[++rear] = value;
}

int dequeue() {
    if (front == -1)
        return -1;
    int value = queue[front++];
    if (front > rear) front = rear = -1;
    return value;
}

void addEdge(int src, int dest) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->vertex = dest;
    newNode->next = adjList[src];
    adjList[src] = newNode;

    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->vertex = src;
    newNode->next = adjList[dest];
    adjList[dest] = newNode;
}

void BFS(int start, int n) {
    for (int i = 0; i < n; i++)
        visited[i] = 0;

    enqueue(start);
    visited[start] = 1;

    printf("BFS Traversal: ");
    while (front != -1) {
        int v = dequeue();
        printf("%d ", v);

        struct Node* temp = adjList[v];
        while (temp != NULL) {
            if (!visited[temp->vertex]) {
                enqueue(temp->vertex);
                visited[temp->vertex] = 1;
            }
            temp = temp->next;
        }
    }
    printf("\n");
}

int main() {
    int n, edges;
    int src, dest, start;

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
        adjList[i] = NULL;

    printf("Enter number of edges: ");
    scanf("%d", &edges);

    printf("Enter edges (u v):\n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &src, &dest);
        addEdge(src, dest);
    }

    printf("Enter starting vertex for BFS: ");
    scanf("%d", &start);

    BFS(start, n);

    return 0;
}
```

Output:

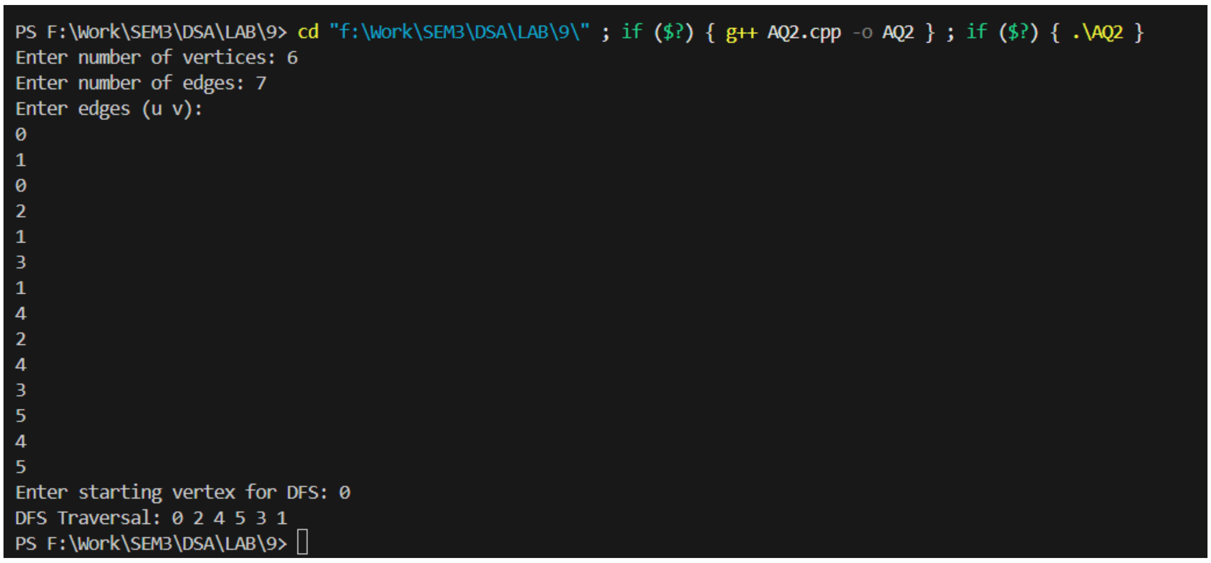


```
PS F:\Work\SEM3\DSA\LAB\9> cd "f:\Work\SEM3\DSA\LAB\9\" ; if ($?) { g++ AQ1.cpp -o AQ1 } ; if ($?) { .\AQ1 }
Enter number of vertices: 6
Enter number of edges: 7
Enter edges (u v):
0
1
0
2
1
3
1
4
2
4
3
5
4
5
Enter starting vertex for BFS: 0
BFS Traversal: 0 2 1 4 3 5
PS F:\Work\SEM3\DSA\LAB\9> 
```

AQ2:

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

struct Node {
    int vertex;
    struct Node* next;
};

struct Node* adjList[MAX];
int visited[MAX];

void addEdge(int src, int dest) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->vertex = dest;
    newNode->next = adjList[src];
    adjList[src] = newNode;

    newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->vertex = src;
    newNode->next = adjList[dest];
    adjList[dest] = newNode;
}

void DFS(int v) {
    visited[v] = 1;
    printf("%d ", v);

    struct Node* temp = adjList[v];
    while (temp != NULL) {
        if (!visited[temp->vertex]) {
            DFS(temp->vertex);
        }
        temp = temp->next;
    }
}

int main() {
    int n, edges;
    int src, dest, start;

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
        adjList[i] = NULL;

    printf("Enter number of edges: ");
    scanf("%d", &edges);

    printf("Enter edges (u v):\n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &src, &dest);
        addEdge(src, dest);
    }

    for (int i = 0; i < n; i++)
        visited[i] = 0;

    printf("Enter starting vertex for DFS: ");
    scanf("%d", &start);

    printf("DFS Traversal: ");
    DFS(start);
    printf("\n");

    return 0;
}
```

Output:

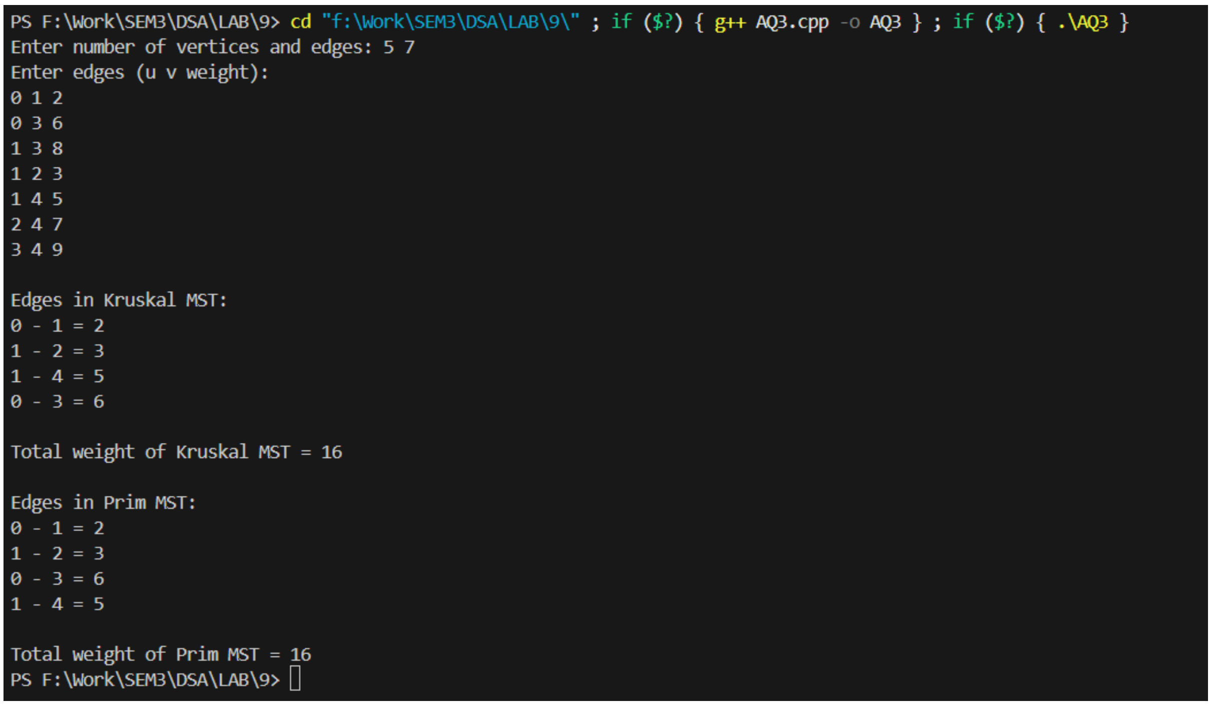


```
PS F:\Work\SEM3\DSA\LAB\9> cd "f:\Work\SEM3\DSA\LAB\9\" ; if ($?) { g++ AQ2.cpp -o AQ2 } ; if ($?) { .\AQ2 }
Enter number of vertices: 6
Enter number of edges: 7
Enter edges (u v):
0
1
0
2
1
3
1
4
2
4
3
5
4
5
Enter starting vertex for DFS: 0
DFS Traversal: 0 2 4 5 3 1
PS F:\Work\SEM3\DSA\LAB\9> 
```

AQ3:

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

struct Edge {
    int u, v, w;
    bool operator<(const Edge& other) const {
        return w < other.w;
    }
};

vector<int> parent, rankSet;
int findSet(int x) {
    if (parent[x] != x)
        parent[x] = findSet(parent[x]);
    return parent[x];
}
void unionSet(int x, int y) {
    x = findSet(x);
    y = findSet(y);
    if (x != y) {
        if (rankSet[x] < rankSet[y]) swap(x, y);
        parent[y] = x;
        if (rankSet[x] == rankSet[y]) rankSet[x]++;
    }
}

int KruskalMST(int V, vector<Edge>& edges) {
    sort(edges.begin(), edges.end());
    parent.resize(V);
    rankSet.assign(V, 0);
    for (int i = 0; i < V; i++) parent[i] = i;

    int mstWeight = 0;
    cout << "\nEdges in Kruskal MST:\n";

    for (auto &e : edges) {
        if (findSet(e.u) != findSet(e.v)) {
            unionSet(e.u, e.v);
            mstWeight += e.w;
            cout << e.u << " - " << e.v << " = " << e.w << "\n";
        }
    }
    return mstWeight;
}

int PrimMST(int V, vector<vector<pair<int,int>>>& adj) {
    vector<int> key(V, INT_MAX), parent(V, -1);
    vector<bool> inMST(V, false);
    key[0] = 0;

    priority_queue<pair<int,int>, vector<pair<int,int>>,
greater<pair<int,int>>> pq;
    pq.push({0, 0});

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        inMST[u] = true;

        for (auto &p : adj[u]) {
            int v = p.first, weight = p.second;
            if (!inMST[v] && weight < key[v]) {
                parent[v] = u;
                key[v] = weight;
                pq.push({key[v], v});
            }
        }
    }

    int mstWeight = 0;
    cout << "\nEdges in Prim MST:\n";
    for (int i = 1; i < V; i++) {
        cout << parent[i] << " - " << i << " = " << key[i] << "\n";
        mstWeight += key[i];
    }
    return mstWeight;
}
```

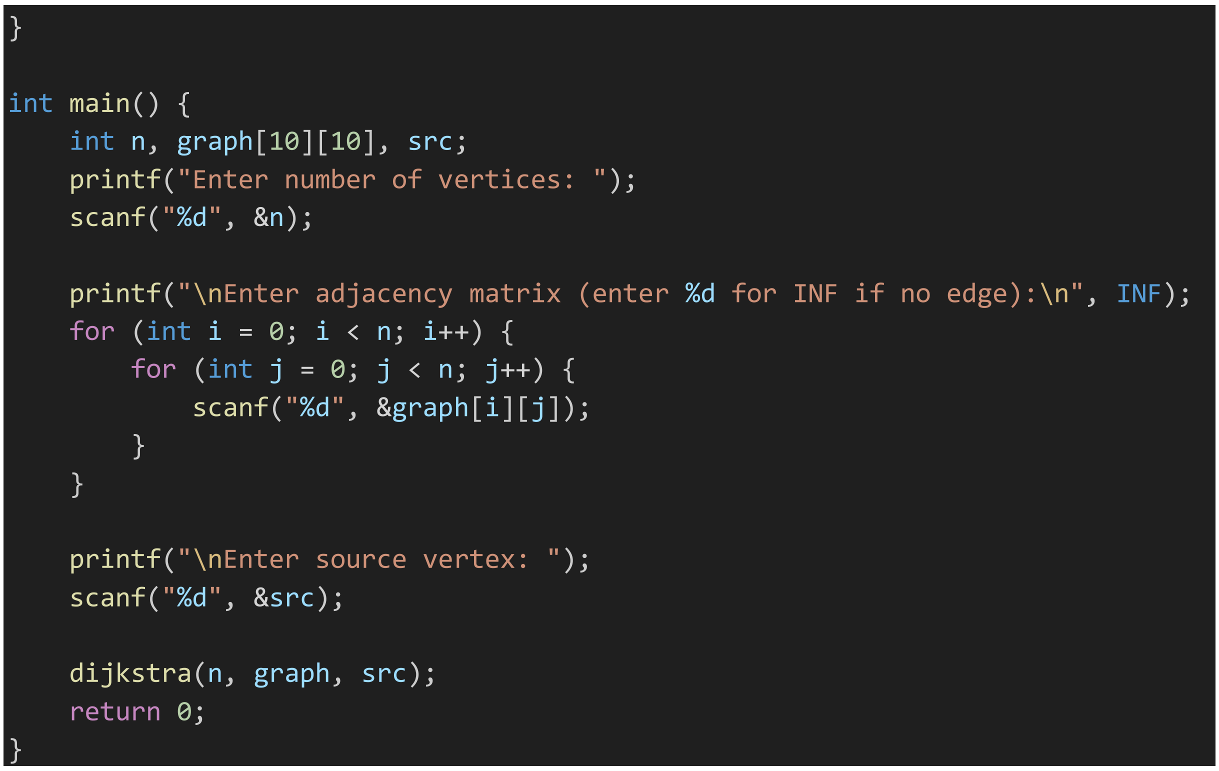
```cpp
int main() {
    int V, E;
    cout << "Enter number of vertices and edges: ";
    cin >> V >> E;

    vector<Edge> edges;
    vector<vector<pair<int,int>>> adj(V);

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

    int kruskalWeight = KruskalMST(V, edges);
    cout << "\nTotal weight of Kruskal MST = " << kruskalWeight << endl;

    int primWeight = PrimMST(V, adj);
    cout << "\nTotal weight of Prim MST = " << primWeight << endl;

    return 0;
}
```
Output:



```
PS F:\Work\SEM3\DSA\LAB\9> cd "f:\Work\SEM3\DSA\LAB\9\" ; if ($?) { g++ AQ3.cpp -o AQ3 } ; if ($?) { .\AQ3 }
Enter number of vertices and edges: 5 7
Enter edges (u v weight):
0 1 2
0 3 6
1 3 8
1 2 3
1 4 5
2 4 7
3 4 9

Edges in Kruskal MST:
0 - 1 = 2
1 - 2 = 3
1 - 4 = 5
0 - 3 = 6

Total weight of Kruskal MST = 16

Edges in Prim MST:
0 - 1 = 2
1 - 2 = 3
0 - 3 = 6
1 - 4 = 5

Total weight of Prim MST = 16
PS F:\Work\SEM3\DSA\LAB\9> 
```

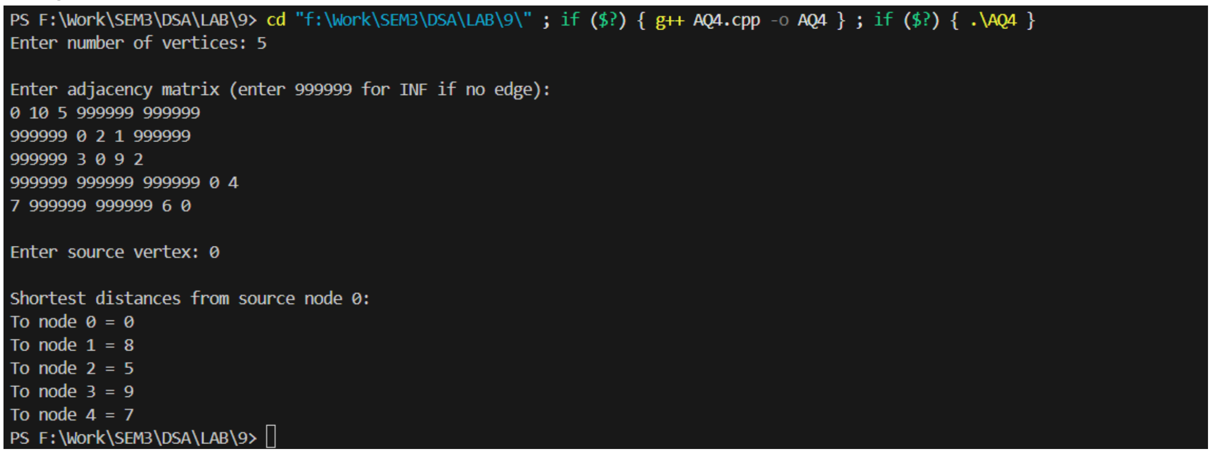
AQ4:

```c
#include <stdio.h>
#define INF 999999

void dijkstra(int n, int graph[10][10], int src) {
    int dist[10], visited[10];
    int i, j, count, minDist, nextNode;

    // Initialize distances & visited array
    for (i = 0; i < n; i++) {
        dist[i] = graph[src][i];
        visited[i] = 0;
    }

    dist[src] = 0;         // Distance to source = 0
    visited[src] = 1;      // Mark source visited
    count = 1;

    // Find shortest paths for all nodes
    while (count < n) {
        minDist = INF;

        // Pick the nearest unvisited vertex
        for (i = 0; i < n; i++) {
            if (!visited[i] && dist[i] < minDist) {
                minDist = dist[i];
                nextNode = i;
            }
        }

        visited[nextNode] = 1;

        // Update distances of adjacent vertices
        for (i = 0; i < n; i++) {
            if (!visited[i]) {
                if (minDist + graph[nextNode][i] < dist[i]) {
                    dist[i] = minDist + graph[nextNode][i];
                }
            }
        }
        count++;
    }

    // Print shortest distances
    printf("\nShortest distances from source node %d:\n", src);
    for (i = 0; i < n; i++) {
        printf("To node %d = %d\n", i, dist[i]);
    }
```



```c
}

int main() {
    int n, graph[10][10], src;
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("\nEnter adjacency matrix (enter %d for INF if no edge):\n", INF);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    printf("\nEnter source vertex: ");
    scanf("%d", &src);

    dijkstra(n, graph, src);
    return 0;
}
```

Output:



```
PS F:\Work\SEM3\DSA\LAB\9> cd "f:\Work\SEM3\DSA\LAB\9\" ; if ($?) { g++ AQ4.cpp -o AQ4 } ; if ($?) { .\AQ4 }
Enter number of vertices: 5

Enter adjacency matrix (enter 999999 for INF if no edge):
0 10 5 999999 999999
999999 0 2 1 999999
999999 3 0 9 2
999999 999999 999999 0 4
7 999999 999999 6 0

Enter source vertex: 0

Shortest distances from source node 0:
To node 0 = 0
To node 1 = 8
To node 2 = 5
To node 3 = 9
To node 4 = 7
PS F:\Work\SEM3\DSA\LAB\9> 
```