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

#define INF 9999

class SelectionSort {
public:
    SelectionSort() {
        cout << "Selection Sort Algorithm\n";
        cout << "--------------------------\n";
        cout << "Enter number of elements: ";
        int n;
        cin >> n;
        vector<int> arr(n);
        cout << "Enter elements:\n";
        for (int i = 0; i < n; i++)
            cin >> arr[i];

        sort(arr);

        cout << "Sorted array:\n";
        for (int i = 0; i < n; i++)
            cout << arr[i] << " ";
        cout << endl;
    }

    void sort(vector<int>& arr) {
        int n = arr.size();
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex])
                    minIndex = j;
            }
            swap(arr[i], arr[minIndex]);
        }
    }
};

class Graph {
public:
    int graph[10][10];
    int n, isDirected, isWeighted;

    Graph() {
```

```
    cout << "Graph Algorithms\n";
    cout << "--------------------------\n";
    cout << "Enter number of vertices: ";
    cin >> n;
    cout << "Is the graph directed? (1 for yes, 0 for no): ";
    cin >> isDirected;
    cout << "Is the graph weighted? (1 for yes, 0 for no): ";
    cin >> isWeighted;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            graph[i][j] = 0;
}

void printGraph() {
    cout << "Adjacency Matrix:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << graph[i][j] << " ";
        cout << endl;
    }
}

void readGraphByEdge() {
    int u, v, w;
    while (true) {
        cout << "Enter edge (u v w) or -1 to stop: ";
        cin >> u;
        if (u == -1)
            break;
        cin >> v >> w;
        graph[u][v] = isWeighted ? w : 1;
        if (!isDirected)
            graph[v][u] = isWeighted ? w : 1;
    }
}

void prims() {
    int cost[10][10];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cost[i][j] = (graph[i][j] == 0) ? INF : graph[i][j];

    int visited[10], dist[10], from[10];
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
        dist[i] = cost[0][i];
        from[i] = 0;
    }

    visited[0] = 1;
    dist[0] = 0;
    int minCost = 0;

    for (int count = 1; count < n; count++) {
        int minDist = INF, v = -1;

        for (int i = 0; i < n; i++) {
            if (!visited[i] && dist[i] < minDist) {
                minDist = dist[i];
                v = i;
            }
        }

        if (v == -1)
            break;

        int u = from[v];
        visited[v] = 1;
        cout << "Edge: " << u << " - " << v << " Weight: " << cost[u][v] << endl;
        minCost += cost[u][v];

        for (int i = 0; i < n; i++) {
            if (!visited[i] && cost[v][i] < dist[i]) {
                dist[i] = cost[v][i];
                from[i] = v;
            }
        }
    }

    cout << "Minimum Cost of MST: " << minCost << endl;
}

void dijkstras() {
    int cost[10][10];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cost[i][j] = (graph[i][j] == 0) ? INF : graph[i][j];

    int visited[10], dist[10], from[10];
    int src = 0;

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

```
            dist[i] = cost[src][i];
            from[i] = (cost[src][i] != INF) ? src : -1;
        }

        visited[src] = 1;
        dist[src] = 0;

        for (int count = 1; count < n; count++) {
            int minDist = INF, v = -1;

            for (int i = 0; i < n; i++) {
                if (!visited[i] && dist[i] < minDist) {
                    minDist = dist[i];
                    v = i;
                }
            }

            if (v == -1)
                break;

            visited[v] = 1;

            for (int i = 0; i < n; i++) {
                if (!visited[i] && dist[i] > dist[v] + cost[v][i]) {
                    dist[i] = dist[v] + cost[v][i];
                    from[i] = v;
                }
            }
        }

        cout << "Shortest paths from source vertex 0:\n";
        for (int i = 0; i < n; i++) {
            if (dist[i] == INF) {
                cout << "To " << i << ": No path\n";
                continue;
            }

            cout << "To " << i << " (Cost: " << dist[i] << "): ";
            vector<int> path;
            for (int j = i; j != -1; j = from[j])
                path.push_back(j);
            reverse(path.begin(), path.end());
            for (int j : path)
                cout << j << (j == i ? "\n" : " -> ");
        }
    }
};
```

```
int main() {
  Graph* g = nullptr;

  while (true) {
    cout << "\n===== MENU =====\n";
    cout << "1. Selection Sort\n";
    cout << "2. Create Graph & Display Adjacency Matrix\n";
    cout << "3. Apply Prim's Algorithm\n";
    cout << "4. Apply Dijkstra's Algorithm\n";
    cout << "5. Exit\n";
    cout << "Enter your choice: ";

    int choice;
    cin >> choice;

    switch (choice) {
      case 1:
        SelectionSort();
        break;

      case 2: {
        g = new Graph();
        g->readGraphByEdge();
        g->printGraph();
        break;
      }

      case 3:
        if (g != nullptr)
          g->prims();
        else
          cout << "Please create a graph first (Option 2).\n";
        break;

      case 4:
        if (g != nullptr)
          g->dijkstras();
        else
          cout << "Please create a graph first (Option 2).\n";
        break;

      case 5:
        cout << "Exiting program.\n";
        delete g; // free memory
        return 0;
        default:
          cout << "Invalid choice. Try again.\n";
      }
    }

    return 0;
}
```