```
I#include <iostream>
#include <vector>
#include <queue>
#include <limits>

using namespace std;

struct Edge {
  int to;
  int weight;
};

struct Node {
  int id;
  int distance;

  bool operator>(const Node& other) const {
    return distance > other.distance;
  }
};

class Graph {
private:
  vector<vector<Edge>> adjList;

public:
  Graph(int n) : adjList(n) {}

  void addEdge(int from, int to, int weight) {
    adjList[from].push_back({to, weight});
  }

  vector<int> dijkstra(int source) const {
    int n = adjList.size();
    vector<int> dist(n, numeric_limits<int>::max());
    vector<bool> visited(n, false);
    priority_queue<Node, vector<Node>, greater<Node>> pq;

    dist[source] = 0;
    pq.push({source, 0}); // {node-int,dist} creates a node

    while (!pq.empty()) {
      Node current = pq.top();
      pq.pop();

      if (visited[current.id]) continue;
      visited[current.id] = true;

      for (const Edge& edge : adjList[current.id]) {
        alt = dist[current.id] + edge.weight;
        if ( alt < dist[edge.to]) {
          dist[edge.to] = alt;
          pq.push({edge.to, dist[edge.to]});
        }
```

```
      }
    }

    return dist;
  }
};

int main() {
  int n, m;
  cout << "Enter number of nodes and edges: ";
  cin >> n >> m;

  Graph graph(n);

  cout << "Enter the edges (format: from to weight):" << endl;
  for (int i = 0; i < m; i++) {
    int from, to, weight;
    cin >> from >> to >> weight;
    graph.addEdge(from, to, weight);
  }

  int source;
  cout << "Enter source node: ";
  cin >> source;

  vector<int> distances = graph.dijkstra(source);
  cout << "Distances from source node " << source << ":" << endl;
  for (int i = 0; i < n; i++) {
    cout << i << ": " << distances[i] << endl;
  }

  return 0;
}
```