Q2.

```
#include <iostream>

#include <vector>

#include <queue>

#include <limits>

using namespace std;

const long long INF = numeric_limits<long long>::max();

struct Edge {

    int to;

    long long weight;

};

void dijkstra(int n, const vector<vector<Edge>>& adj) {

    vector<long long> dist(n + 1, INF);

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

    dist[1] = 0;

    pq.push({0, 1});

    while (!pq.empty()) {

        long long d = pq.top().first;

        int u = pq.top().second;

        pq.pop()

        if (d > dist[u]) continue;

        for (const Edge& edge : adj[u]) {

            int v = edge.to;

            long long weight = edge.weight;

            if (dist[u] + weight < dist[v]) {

                dist[v] = dist[u] + weight;
```

```
                pq.push({dist[v], v});
            }
        }
    }
    for (int i = 1; i <= n; ++i) {
        cout << dist[i] << " ";
    }
    cout << endl;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;

    vector<vector<Edge>> adj(n + 1);

    for (int i = 0; i < m; ++i) {
        int a, b;
        long long c;
        cin >> a >> b >> c;
        adj[a].push_back({b, c});
    }
```

```
    dijkstra(n, adj);

    return 0;

}
```

Explanation :

- The adjacency list adj is built from the input where each city has a list of its outgoing edges.
- Dijkstra's Algorithm:
    - We initialize distances with infinity and set the distance to the source (Syrjälä) to 0.
    - We use a priority queue to explore the shortest paths, updating distances and pushing updated distances to the priority queue.