1.

(a) We create an adjacency matrix of (n+1) * (n+1) where n is the number of vertices. It iterates over the given edges and assigns the corresponding weight to the appropriate position in the adjacency matrix.

(b) Here, we create an adjacency list representation of the graph by iterating over the edges and appending the destination vertex and weight as a tuple to the list corresponding to the source vertex.

2. We perform a Breadth-first Search (BFS) traversal on the given graph starting from city 1. We do this by using a queue and visiting each neighbouring city in the order they are encountered marking them as visited to avoid cycles.

3. This solution performs a Depth First Search (DFS) traversal on the given graph representation of the cities and roads, starting from city 1. It does this by recursively exploring each neighbouring city in sorted order (for consistency in output), marking them as visited to avoid cycles and appending them to path.

4. The solution detects the cycle by performing DFS with additional tracking of visited and recursion stack flags. It checks if a visited node is encountered while being in the current stack.

5. Here, we are using BFS and maintaining a queue of nodes along with their time and path to the shortest distance and minimum time.

6. Here we use DFS to find the maximum number of diamonds ~~that can be~~ by keeping track of visited cells and counting the diamonds encountered. Then ultimately selecting the max diamond count from all possible positions.