QN 1 (A) BRIEF EXPLANATION

We use an **adjacency matrix** because it provides **efficient distance lookups** between cities. Here’s why:

✅ **Fast Access:**

* Since it's a **2D list**, we can instantly retrieve the distance between any two cities in **O(1) time** using graph[i][j].
* Example: To get the distance between **City 2 and City 5**, we simply check graph[1][4] (since indices start at 0).

✅ **Simple Implementation:**

* The matrix is **easy to construct** and understand. Each row and column directly represents a city.
* There's **no need for complex data structures** like linked lists or hash maps.

✅ **Good for Small Graphs:**

* Since the given TSP problem has **only 7 cities**, using a **7×7 matrix (49 elements)** is **memory-efficient**.
* For **larger graphs**, adjacency lists would be better, but for **small TSP instances, the matrix is ideal**.

QN 1 (B)

### ****Traveling Salesman Problem (TSP) Objective****

The **goal of the Traveling Salesman Problem (TSP)** is to find the **shortest possible route** that:  
1️⃣ **Starts from a given city** (City 1 in our case).  
2️⃣ **Visits each of the other cities exactly once** (No revisiting).  
3️⃣ **Returns to the starting city** after visiting all cities.  
4️⃣ **Minimizes the total travel distance**.

### ****Assumptions Made****

✅ **Symmetric TSP** – The distances between cities are **the same in both directions** (e.g., the distance from **City 2 to City 5** is **12**, the same as from **City 5 to City 2**).

✅ **No Missing Cities** – The graph is **fully connected**, meaning every city is reachable from others (even if not directly).

✅ **Integer Distances** – The distances are **fixed whole numbers**, meaning we don’t deal with floating-point precision issues.

✅ **Single Salesman** – We assume **one person** is traveling (no multiple salesmen or multiple routes).

CLASSICAL TSP SOLUTION

1. Nearest Neighbor
2. Implementation
3. Result : “final tour from code”

SELF-ORGANIZING MAP APPROACH