## **(a). Graph Representation**

## Chosen Data Structure: **An Adjacency Matrix**

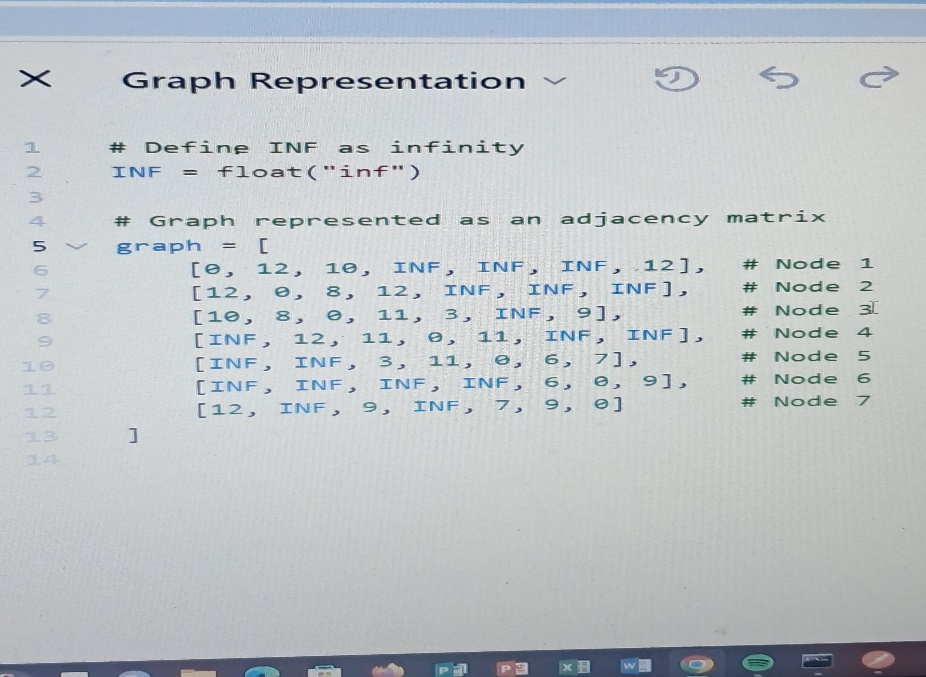
An **Adjacency Matrix** is a **2D array** used to represent a graph, where rows and columns correspond to cities (nodes). Each entry Matrix[i][j] stores the weight (distance) between city i and city j, with 0 for the same city (i=j), the actual distance if a direct route exists, and ∞ (or a large value) if no direct connection exists.

#### **2. Adjacency Matrix for the Given Graph** Here’s how the matrix looks:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **1** | 0 | 12 | 10 | ∞ | ∞ | ∞ | 12 |
| **2** | 12 | 0 | 8 | 12 | ∞ | ∞ | ∞ |
| **3** | 10 | 8 | 0 | 11 | 3 | ∞ | 9 |
| **4** | ∞ | 12 | 11 | 0 | 11 | ∞ | ∞ |
| **5** | ∞ | ∞ | 3 | 11 | 0 | 10 | 7 |
| **6** | ∞ | ∞ | ∞ | ∞ | 10 | 0 | 9 |
| **7** | 12 | ∞ | 9 | ∞ | 7 | 9 | 0 |

* **Diagonal values are 0** (distance from a city to itself).
* **"∞" represents no direct route between cities.**

#### **Python Code for Adjacency Matrix Representation**

****Here's how you can implement this in **Python**:

**Justification of Adjacency Matrix**

The **Adjacency Matrix** is chosen because it allows **O(1) time** retrieval of distances using **graph[i][j]**, ensuring fast lookups for the TSP problem. It requires **O(N²) memory**, which is efficient for small, fully connected graphs. Additionally, its **simple 2D list structure** makes it easy to implement and understand.