1. Introduction and Biological Inspiration

The algorithm implemented in this project, **Ant Colony Optimization (ACO)**, is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. It is bio-inspired, modeled after the behavior of real ant colonies.

In nature, ants wander randomly, and upon finding food, return to their colony while laying down **pheromone trails**. If other ants find such a path, they are likely not to keep traveling at random, but instead to follow the trail, reinforcing it if they eventually find food. Over time, the pheromone trail starts to evaporate, reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones.

This project applies this "swarm intelligence" to the **Traveling Salesperson Problem (TSP)**, where artificial "ants" construct solutions by moving from city to city on a graph.

2. Problem Formulation and Data Structures

2.1 The Environment

The problem consists of **N** cities. We represent the environment using two primary matrices.

1. **Distance Matrix (D):** First, we calculate the Euclidean distance between every pair of cities. Since the distance from City A to City B is the same as from City B to City A, this matrix is **symmetric**.
   `D[i][j] = D[j][i]`
   The main diagonal consists of zeros (distance from a city to itself is 0). While a full **N x N** matrix is often used for implementation ease, mathematically, the unique information required is only the upper triangle, determined by the formula:
   `Unique Edges = N * (N - 1) / 2`
2. **Visibility Matrix (Eta):** To aid the ants in making "locally good" decisions (heuristics), we pre-calculate a visibility matrix. The visibility, denoted as **Eta**, is defined as the inverse of the distance:
   `Eta[i][j] = 1 / Distance[i][j]`
   This constant matrix allows the algorithm to perform faster computations during the iterative process, favoring closer cities by default.

3. The Probabilistic Decision Process

Unlike a "Greedy Algorithm" which always chooses the absolute closest city, an ant in ACO chooses the next city **probabilistically**. This stochastic nature is crucial as it allows the algorithm to escape local optima (sub-optimal solutions that look good locally).

3.1 Transition Probability Formula

An ant currently at city **i** chooses to move to city **j** based on the following probability formula.

The probability **P** is calculated as:

```
P_ij = ( (Tau_ij ^ Alpha) * (Eta_ij ^ Beta) ) / ( Sum of all available paths )
```

Where:

- **Tau (Pheromone):** The amount of pheromone on the edge between i and j. This represents the "memory" of the colony—the "prejudice" passed down from ancestors indicating that this path was successful in the past.
- **Eta (Visibility):** The heuristic value (1 / distance). This represents the immediate attractiveness of the move.
- **Alpha:** The parameter controlling the influence of the **pheromone**.
    - *High Alpha:* Ants rely heavily on "ancestral knowledge." The system converges quickly but may get stuck in bad solutions (stagnation).
- **Beta:** The parameter controlling the influence of **visibility**.
    - *High Beta:* Ants behave like a greedy algorithm, only caring about the immediate distance. The colony may fail to find complex, global shortcuts.

3.2 The Roulette Wheel Selection

The algorithm calculates these probability scores for all unvisited cities. Even if City A has a 90% probability and City B has 10%, the ant *might* still choose City B. This randomness ensures exploration of the search space.

4. Pheromone Update Mechanism

Once all ants have completed their tours, the system updates the "global memory" (the pheromone matrix). This process involves two distinct phases: **Evaporation** and **Deposition**.

4.1 Phase 1: Evaporation

Before new pheromones are added, the existing trails must decay. This simulates nature (pheromones vanishing over time) and prevents the algorithm from converging too early to a poor solution.

Mathematically, this is a **Scalar Matrix Multiplication**. If **T** is the Pheromone Matrix and **Rho** is the evaporation rate (e.g., 0.83), the operation is:

```
T_new = T_old * (1 - Rho)
```

This means every single value in the matrix is multiplied by `(1 - 0.83)`, or `0.17`.

- **Effect of Rho:**

- If Rho is close to 1, pheromones evaporate rapidly. The colony "forgets" quickly and behaves randomly.
- If Rho is close to 0, old trails persist indefinitely, limiting new exploration.

4.2 Phase 2: Pheromone Deposition

After evaporation, successful ants add new pheromones to the edges they traversed. The amount of pheromone deposited is inversely proportional to the total length of the tour found by the ant.

```
Delta_Tau = Q / Tour_Length
```

Where:

- **Q:** A constant weight factor (pheromone deposit constant).
- **Tour_Length:** The total distance traveled by that specific ant.

The update equation for the pheromone matrix becomes:

```
Tau_new = Tau_current + Sum(Delta_Tau)
```

By adding `Q / Length`, shorter tours result in larger values of `Delta_Tau`. Consequently, edges belonging to shorter paths receive more reinforcement. In future iterations, the colony's "prejudice" will shift toward these optimized paths, increasing the probability that future generations will select them.

5. Summary of Algorithm Workflow

1. **Initialization:** Compute Distance and Visibility matrices. Initialize Pheromone matrix with a base value of 1.0.
2. **Construction:** Place **m** ants on random cities.
3. **Traversal:** Each ant builds a tour by probabilistically choosing the next city using the Alpha and Beta weighted formula.
4. **Evaluation:** Calculate the total length of each ant's tour.
5. **Feedback (Update):**
    - Multiply the entire Pheromone Matrix by `(1 - Rho)` (Evaporation).
    - Add `Q / Length` to the edges visited by the ants (Deposition).
6. **Termination:** Repeat steps 2-5 for the maximum number of iterations or until the system converges.