9. Write the python to implement Travelling Salesman Problem

**AIM :** to implement Travelling Salesman Problem

**ALGORITHM :**

1. Create a graph where cities are represented as nodes and the distances between cities are represented as edges.
2. Start from any city as the initial city.
3. Use a method to generate permutations of cities to visit. This could be done using a recursive function or an iterative approach.
4. For each permutation:

* Calculate the total distance of the route that visits cities in the order of the permutation and returns to the starting city.
* If this distance is shorter than the previously found shortest distance, update the shortest distance and record the permutation.

1. After iterating through all permutations, the recorded permutation with the shortest distance is the optimal solution.

**PROGRAM :**

import itertools

def calculate\_distance(city1, city2):

return ((city1[0] - city2[0]) \*\* 2 + (city1[1] - city2[1]) \*\* 2) \*\* 0.5

def total\_distance(path, cities):

distance = 0

for i in range(len(path) - 1):

distance += calculate\_distance(cities[path[i]], cities[path[i + 1]])

distance += calculate\_distance(cities[path[-1]], cities[path[0]]) # Return to starting city

return distance

def brute\_force\_tsp(cities):

num\_cities = len(cities)

all\_permutations = list(itertools.permutations(range(num\_cities)))

min\_distance = float('inf')

best\_path = []

for perm in all\_permutations:

distance = total\_distance(perm, cities)

if distance < min\_distance:

min\_distance = distance

best\_path = perm

return best\_path, min\_distance

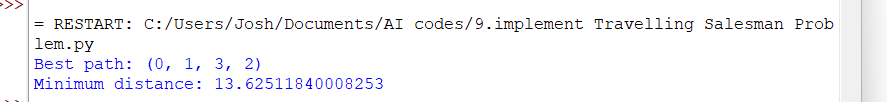
cities = [(0, 0), (2, 4), (3, 1), (5, 3)]

best\_path, min\_distance = brute\_force\_tsp(cities)

print("Best path:", best\_path)

print("Minimum distance:", min\_distance)

**OUT PUT :**

****