

▼ CSP

BY: Vivek Vittal Biragoni, 211AI041

19/05/2023

```
def tsp(adj_matrix, city_names):
    num_cities = len(city_names)
    visited = [False] * num_cities
    path = []

    # Start from the first city
    current_city = 0
    visited[current_city] = True
    path.append(city_names[current_city])

    # Visit the remaining cities
    for _ in range(num_cities - 1):
        nearest_city = None
        min_distance = float('inf')

        # Find the nearest unvisited city
        for neighbor_city in range(num_cities):
            if not visited[neighbor_city] and adj_matrix[current_city][neighbor_city] < min_distance:
                nearest_city = neighbor_city
                min_distance = adj_matrix[current_city][neighbor_city]


        # Mark the nearest city as visited and add it to the path
        visited[nearest_city] = True
        path.append(city_names[nearest_city])
        current_city = nearest_city

    # Return to the starting city
    path.append(city_names[0])

    return path

# Example usage
adj_matrix = [
    [0, 15, 13, 16, 12],
    [15, 0, 14, 11, 17],
    [13, 14, 0, 19, 18],
    [16, 11, 19, 0, 14],
    [12, 17, 18, 14, 0]
]
city_names = ["Arad", "Bucharest", "Craiova", "Dobreta", "Eforie"]

tsp_path = tsp(adj_matrix, city_names)
print("TSP path:", tsp_path)
```

 TSP path: ['Arad', 'Eforie', 'Dobreta', 'Bucharest', 'Craiova', 'Arad']

▼ NOTES:

The algorithm used in the provided code is a greedy algorithm called the nearest neighbor heuristic. It starts from a randomly selected city and repeatedly selects the nearest unvisited city until all cities have been visited.

The algorithm maintains a list of visited cities and a path that represents the order in which the cities are visited. It initializes the current city as the starting city, marks it as visited, and appends it to the path.

In each iteration, the algorithm looks for the nearest unvisited city to the current city by comparing the distances in the adjacency matrix. It selects the city with the minimum distance and updates the current city to the nearest city. The nearest city is then marked as visited and added to the path.

This process continues until all cities have been visited. Finally, the algorithm adds the starting city back to the path to complete the tour.

The nearest neighbor heuristic is a simple and efficient approach to approximate the solution for the TSP. However, it does not guarantee finding the optimal solution.

```

from sklearn.cluster import KMeans
import numpy as np

def minimize_squared_distances(adj_matrix, city_names, num_airports):
    # Convert the adjacency matrix to a distance matrix
    distance_matrix = np.sqrt(adj_matrix)

    # Perform k-means clustering to find the optimal airport locations
    kmeans = KMeans(n_clusters=num_airports, random_state=0)
    kmeans.fit(distance_matrix)

    # Find the nearest airport for each city
    nearest_airports = kmeans.predict(distance_matrix)

    # Calculate the sum of squared distances from each city to its nearest airport
    sum_squared_distances = 0
    for city_idx, city_name in enumerate(city_names):
        nearest_airport_idx = nearest_airports[city_idx]
        nearest_airport_distance = distance_matrix[city_idx][nearest_airport_idx]
        sum_squared_distances += nearest_airport_distance ** 2

    # Get the coordinates of the airport locations
    airport_locations = kmeans.cluster_centers_

    return sum_squared_distances, airport_locations

# Example usage
num_airports = 3

sum_squared_distances, airport_locations = minimize_squared_distances(adj_matrix, city_names, num_airports)
print("Sum of squared distances:", sum_squared_distances)
print("Airport locations:", airport_locations)

Sum of squared distances: 59.0
Airport locations: [[1.73205081 3.99804449 3.92409598 3.87082869 1.73205081]
 [3.60555128 3.74165739 0. 4.35889894 4.24264069]
 [3.93649167 1.6583124 4.05027817 1.6583124 3.93238151]]
C:\Users\vivek\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packag
warnings.warn(

```

We want to find the best locations for three new airports on a map. We have a list of cities and their distances from each other. We will use a special method called K-means clustering to help us.

First, we will convert the list of distances into a different kind of list called a distance matrix. Then, we will use K-means clustering to find the best places to put the airports. K-means clustering is like grouping similar things together.

After finding the airport locations, we will calculate the sum of squared distances from each city to its nearest airport. This helps us measure how close each city is to its nearest airport.

In the end, we will get two things: the sum of squared distances, which tells us how good our airport placements are, and the coordinates of the airport locations on the map.

By using this method, we can find the best places to put the airports and make sure that cities are as close as possible to their nearest airport.

Double-click (or enter) to edit

