DESIGN AND ANALYSIS OF ALGORITHM

PRACTICAL-2

NAME: VEDANT BHUTADA

ROLL: 69

BATCH: A4

Aim: Construction of Minimum Spanning Tree

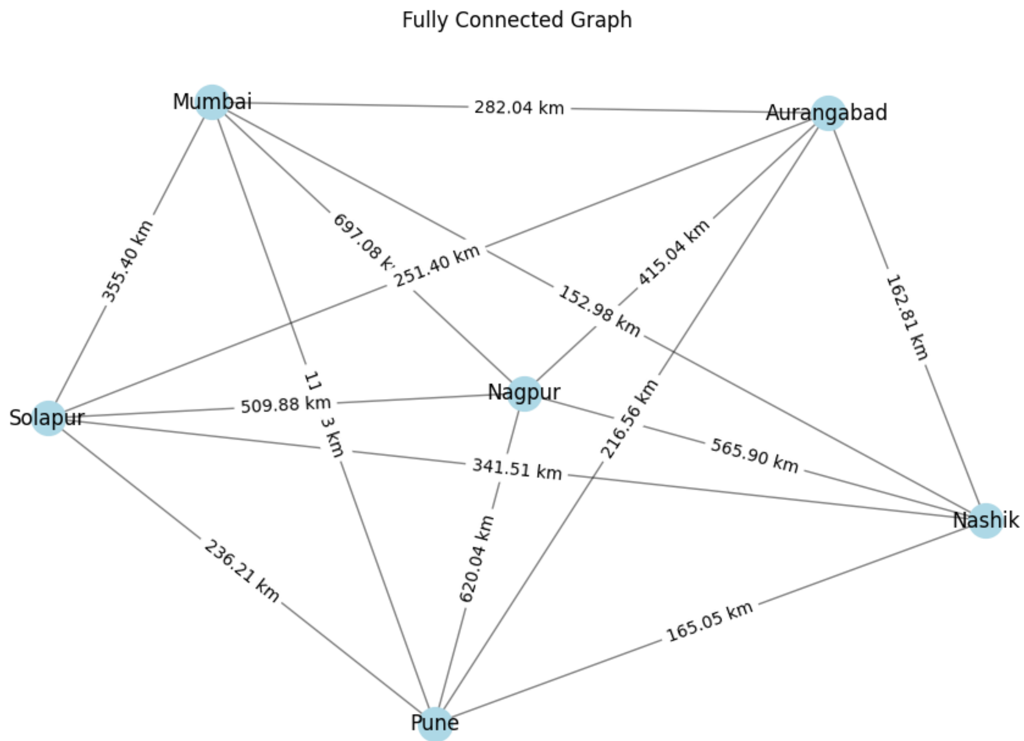
Problem Statement: A telecommunications organization has offices spanned across multiple locations around the globe. It has to use leased phone lines for connecting all these offices with each other. The organization, wants to use minimum cost for connecting all its offices. This requires that all the offices should be connected using a minimum number of leased lines so as to reduce the effective cost. A. Consider the following for deciding connections in same state in India: i. Find the latitude and longitude of cities in same state. Consider 4 to 6 cities. ii. Calculate the cost of connecting each pair of offices by computing the distance between different pair of different cities (as considered in part A) and construct a fully connected graph. iii. Compute a minimum spanning tree using either Prims or Kruskals Method to find the cost of connecting offices in different cities.

6 CITIES

```
from geopy.geocoders import Nominatim
from geopy.distance import geodesic
from itertools import combinations
import networkx as nx
import matplotlib.pyplot as plt
import time

cities = ["Mumbai", "Pune", "Nagpur", "Nashik", "Aurangabad", "Solapur"]

city_coordinates = {}

geolocator = Nominatim(user_agent="telecom_office_connection")

for city in cities:
    location = geolocator.geocode(city)
    if location:
        city_coordinates[city] = (location.latitude, location.longitude)
    else:
        print(f"Location not found for {city}")
edges = []
for city1, city2 in combinations(cities, 2):
    distance = geodesic(city_coordinates[city1], city_coordinates[city2]).kilometers
    edges.append((city1, city2, distance))
edges.sort(key=lambda x: x[2])
mst= nx.Graph()
def find(parent, node):
    if parent[node] == node:
        return node
    return find(parent, parent[node])

parent = {city: city for city in cities}

# Kruskal's Algo
start_time = time.time()

for edge in edges:
    city1, city2, weight = edge
    root1 = find(parent, city1)
    root2 = find(parent, city2)

    if root1 != root2:
        mst.add_edge(city1, city2, weight=weight)
        parent[root1] = root2

end_time = time.time()
execution_time = end_time - start_time

print("Minimum Spanning Tree Edges:")
for edge in mst.edges:
    city1, city2 = edge
    cost = mst[city1][city2]['weight']
    print(f"{city1} - {city2}: {cost} km")
print(f"Time taken to compute MST: {execution_time:.4f} seconds")

import matplotlib.pyplot as plt
```

```
#fully connected graph
fully_connected_graph = nx.Graph()

for edge in edges:
    city1, city2, weight = edge
    fully_connected_graph.add_edge(city1, city2, weight=weight)

# Plot
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(fully_connected_graph, seed=42)
labels = {city: city for city in fully_connected_graph.nodes()}

nx.draw_networkx_nodes(fully_connected_graph, pos, node_size=400, node_color='lightblue')
nx.draw_networkx_edges(fully_connected_graph, pos, edgelist=fully_connected_graph.edges(), width=1.0, alpha=0.5)
nx.draw_networkx_labels(fully_connected_graph, pos, labels, font_size=12, font_color='black')

# Respective costs
edge_labels = {(city1, city2): f"{fully_connected_graph[city1][city2]['weight']:.2f} km" for city1, city2 in fully_connected_graph.edges(
nx.draw_networkx_edge_labels(fully_connected_graph, pos, edge_labels=edge_labels)

# Show plot
plt.title("Fully Connected Graph")
plt.axis('off')
plt.show()


# Total cost
total_cost = sum([mst[city1][city2]['weight'] for city1, city2 in mst.edges])

print(f"Total Cost of Connecting Offices: {total_cost} km")

#OUTPUT
```

```
Minimum Spanning Tree Edges:
Mumbai - Pune: 119.22916693804272 km
Mumbai - Nashik: 152.97575128782637 km
Pune - Solapur: 236.2120988316117 km
Nashik - Aurangabad: 162.80953160402416 km
Aurangabad - Nagpur: 415.03512074548814 km
Time taken to compute MST: 0.0003 seconds
```




```
Total Cost of Connecting Offices: 1086.2616694069932 km
```

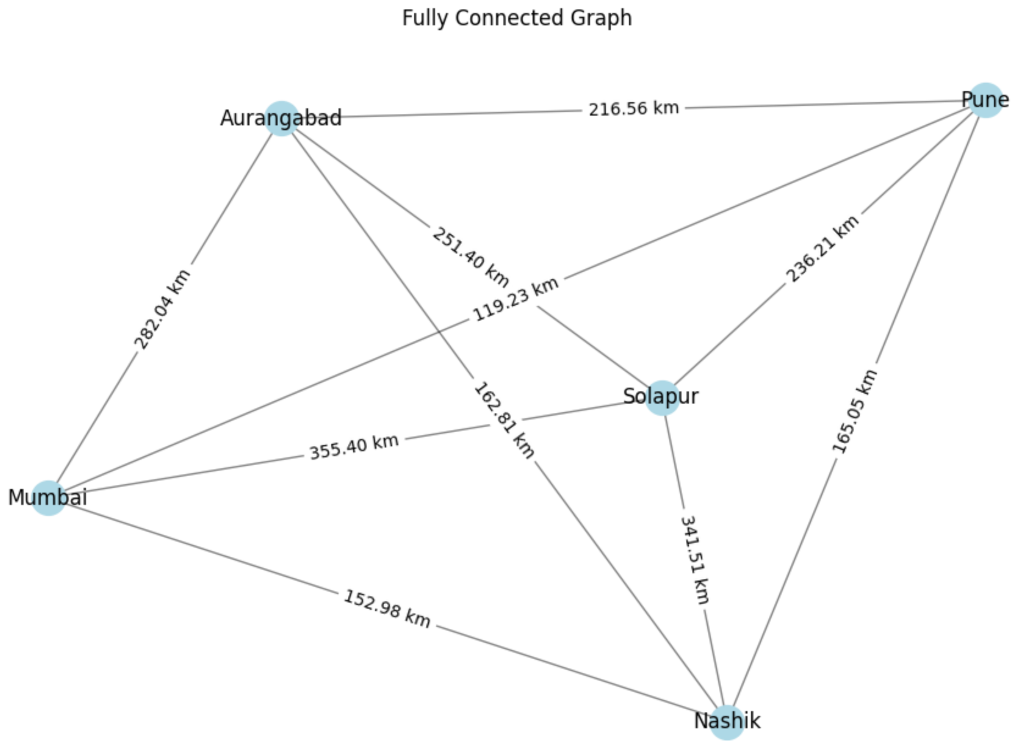
5 CITIES

```
from geopy.geocoders import Nominatim
from geopy.distance import geodesic
```

```
from itertools import combinations
import networkx as nx
import matplotlib.pyplot as plt
import time

cities = ["Mumbai", "Pune", "Nashik", "Aurangabad", "Solapur"]

city_coordinates = {}

geolocator = Nominatim(user_agent="telecom_office_connection")

for city in cities:
    location = geolocator.geocode(city)
    if location:
        city_coordinates[city] = (location.latitude, location.longitude)
    else:
        print(f"Location not found for {city}")
edges = []
for city1, city2 in combinations(cities, 2):
    distance = geodesic(city_coordinates[city1], city_coordinates[city2]).kilometers
    edges.append((city1, city2, distance))
edges.sort(key=lambda x: x[2])
mst= nx.Graph()
def find(parent, node):
    if parent[node] == node:
        return node
    return find(parent, parent[node])

parent = {city: city for city in cities}

# Kruskal's Algo
start_time = time.time()

for edge in edges:
    city1, city2, weight = edge
    root1 = find(parent, city1)
    root2 = find(parent, city2)

    if root1 != root2:
        mst.add_edge(city1, city2, weight=weight)
        parent[root1] = root2

end_time = time.time()
execution_time = end_time - start_time

print("Minimum Spanning Tree Edges:")
for edge in mst.edges:
    city1, city2 = edge
    cost = mst[city1][city2]['weight']
    print(f"{city1} - {city2}: {cost} km")
print(f"Time taken to compute MST: {execution_time:.4f} seconds")

import matplotlib.pyplot as plt

#fully connected graph
fully_connected_graph = nx.Graph()

for edge in edges:
    city1, city2, weight = edge
    fully_connected_graph.add_edge(city1, city2, weight=weight)

# Plot
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(fully_connected_graph, seed=42)
labels = {city: city for city in fully_connected_graph.nodes()}

nx.draw_networkx_nodes(fully_connected_graph, pos, node_size=400, node_color='lightblue')
nx.draw_networkx_edges(fully_connected_graph, pos, edgelist=fully_connected_graph.edges(), width=1.0, alpha=0.5)
nx.draw_networkx_labels(fully_connected_graph, pos, labels, font_size=12, font_color='black')

# Respective costs
edge_labels = {(city1, city2): f"{fully_connected_graph[city1][city2]['weight']:.2f} km" for city1, city2 in fully_connected_graph.edges(
nx.draw_networkx_edge_labels(fully_connected_graph, pos, edge_labels=edge_labels)

# Show plot
plt.title("Fully Connected Graph")
plt.axis('off')
plt.show()


# Total cost
total_cost = sum([mst[city1][city2]['weight'] for city1, city2 in mst.edges])
```

```
print(f"Total Cost of Connecting Offices: {total_cost} km")


#OUTPUT
```

```
Minimum Spanning Tree Edges:
Mumbai - Pune: 119.22916693804272 km
Mumbai - Nashik: 152.97575128782637 km
Pune - Solapur: 236.2120988316117 km
Nashik - Aurangabad: 162.80953160402416 km
Time taken to compute MST: 0.0002 seconds
```



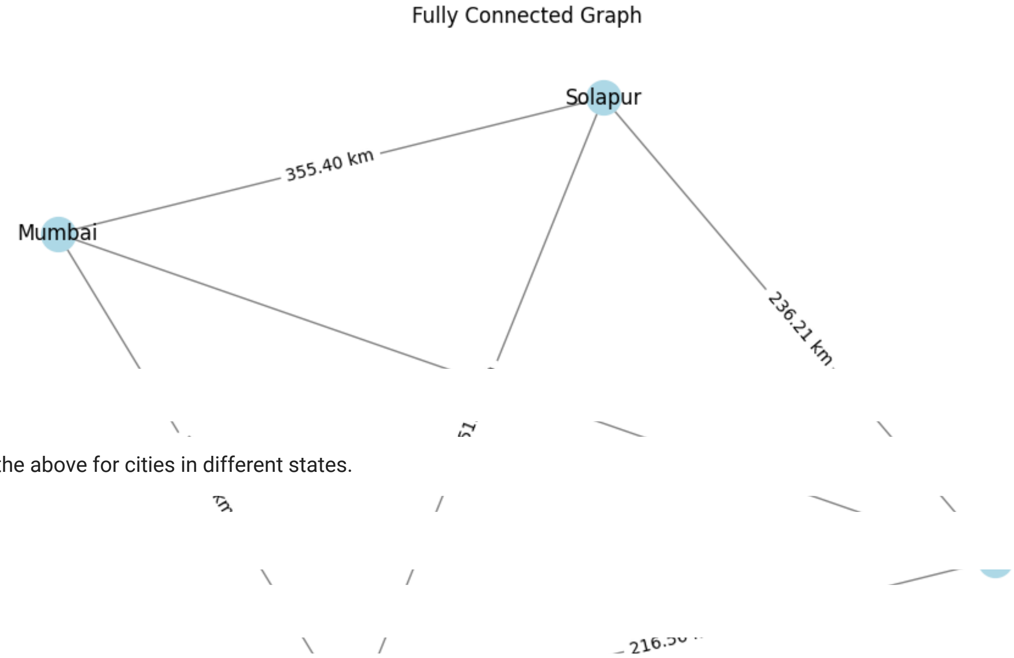

```
Total Cost of Connecting Offices: 671.226548661505 km
```

4 CITIES

```
from geopy.geocoders import Nominatim
from geopy.distance import geodesic
from itertools import combinations
import networkx as nx
import matplotlib.pyplot as plt
import time

cities = ["Mumbai", "Pune", "Aurangabad", "Solapur"]

city_coordinates = {}

geolocator = Nominatim(user_agent="telecom_office_connection")

for city in cities:
    location = geolocator.geocode(city)
    if location:
        city_coordinates[city] = (location.latitude, location.longitude)
    else:
        print(f"Location not found for {city}")
edges = []
for city1, city2 in combinations(cities, 2):
    distance = geodesic(city_coordinates[city1], city_coordinates[city2]).kilometers
    edges.append((city1, city2, distance))
edges.sort(key=lambda x: x[2])
mst= nx.Graph()
def find(parent, node):
    if parent[node] == node:
        return node
    return find(parent, parent[node])
```

```
parent = {city: city for city in cities}

# Kruskal's Algo
start_time = time.time()

for edge in edges:
    city1, city2, weight = edge
    root1 = find(parent, city1)
    root2 = find(parent, city2)

    if root1 != root2:
        mst.add_edge(city1, city2, weight=weight)
        parent[root1] = root2

end_time = time.time()
execution_time = end_time - start_time

print("Minimum Spanning Tree Edges:")
for edge in mst.edges:
    city1, city2 = edge
    cost = mst[city1][city2]['weight']
    print(f"{city1} - {city2}: {cost} km")
print(f"Time taken to compute MST: {execution_time:.4f} seconds")

import matplotlib.pyplot as plt

#fully connected graph
fully_connected_graph = nx.Graph()

for edge in edges:
    city1, city2, weight = edge
    fully_connected_graph.add_edge(city1, city2, weight=weight)

# Plot
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(fully_connected_graph, seed=42)
labels = {city: city for city in fully_connected_graph.nodes()}

nx.draw_networkx_nodes(fully_connected_graph, pos, node_size=400, node_color='lightblue')
nx.draw_networkx_edges(fully_connected_graph, pos, edgelist=fully_connected_graph.edges(), width=1.0, alpha=0.5)
nx.draw_networkx_labels(fully_connected_graph, pos, labels, font_size=12, font_color='black')

# Respective costs
edge_labels = {(city1, city2): f"{fully_connected_graph[city1][city2]['weight']:.2f} km" for city1, city2 in fully_connected_graph.edges(
nx.draw_networkx_edge_labels(fully_connected_graph, pos, edge_labels=edge_labels)

# Show plot
plt.title("Fully Connected Graph")
plt.axis('off')
plt.show()


# Total cost
total_cost = sum([mst[city1][city2]['weight'] for city1, city2 in mst.edges])

print(f"Total Cost of Connecting Offices: {total_cost} km")


#OUTPUT
```

```
WARNING:urllib3.connectionpool:Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=None)) aft
Minimum Spanning Tree Edges:
Mumbai - Pune: 119.22916693804272 km
Pune - Aurangabad: 216.56124085661867 km
Pune - Solapur: 236.2120988316117 km
Time taken to compute MST: 0.0001 seconds
```



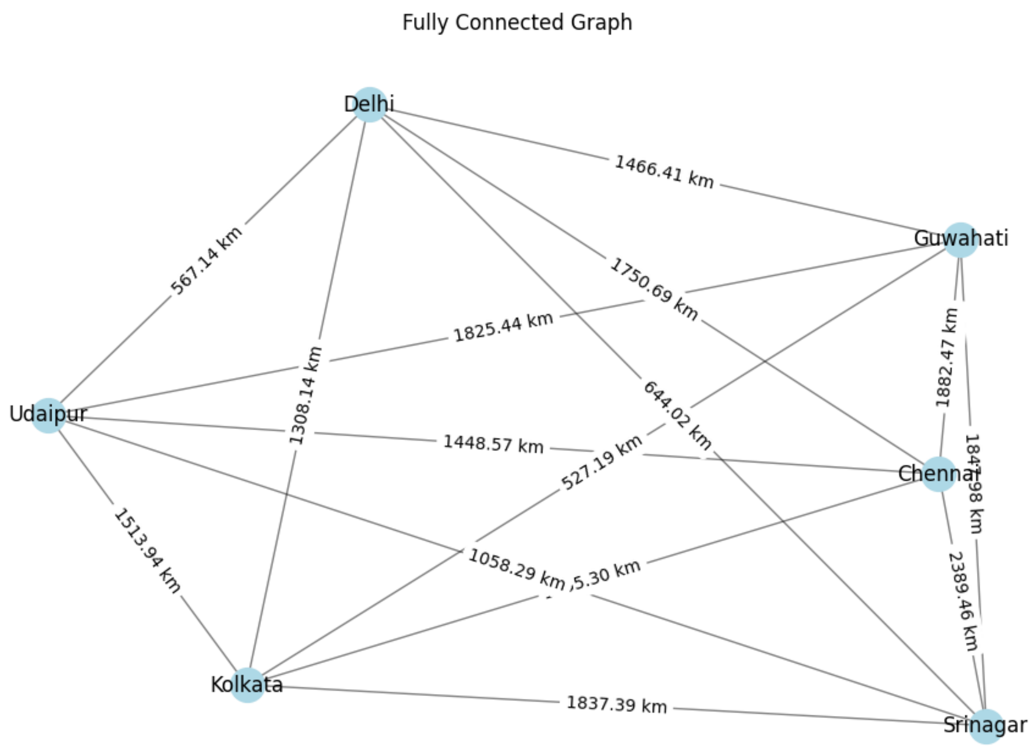
B. Repeat the above for cities in different states.

6 CITIES

```python
from geopy.geocoders import Nominatim
from geopy.distance import geodesic
from itertools import combinations
import networkx as nx
import matplotlib.pyplot as plt
import time

cities = ["Delhi", "Kolkata", "Chennai", "Guwahati", "Udaipur", "Srinagar"]

city_coordinates = {}

geolocator = Nominatim(user_agent="telecom_office_connection")

for city in cities:
    location = geolocator.geocode(city)
    if location:
        city_coordinates[city] = (location.latitude, location.longitude)
    else:
        print(f"Location not found for {city}")
edges = []
for city1, city2 in combinations(cities, 2):
    distance = geodesic(city_coordinates[city1], city_coordinates[city2]).kilometers
    edges.append((city1, city2, distance))
edges.sort(key=lambda x: x[2])
mst= nx.Graph()
def find(parent, node):
    if parent[node] == node:
        return node
    return find(parent, parent[node])

parent = {city: city for city in cities}

# Kruskal's Algo
start_time = time.time()

for edge in edges:
    city1, city2, weight = edge
    root1 = find(parent, city1)
    root2 = find(parent, city2)

    if root1 != root2:
        mst.add_edge(city1, city2, weight=weight)
        parent[root1] = root2

end_time = time.time()
execution_time = end_time - start_time

print("Minimum Spanning Tree Edges:")
for edge in mst.edges:
    city1, city2 = edge
```

```
    cost = mst[city1][city2]['weight']
    print(f"{city1} - {city2}: {cost} km")
print(f"Time taken to compute MST: {execution_time:.4f} seconds")

import matplotlib.pyplot as plt

#fully connected graph
fully_connected_graph = nx.Graph()

for edge in edges:
    city1, city2, weight = edge
    fully_connected_graph.add_edge(city1, city2, weight=weight)

# Plot
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(fully_connected_graph, seed=42)
labels = {city: city for city in fully_connected_graph.nodes()}

nx.draw_networkx_nodes(fully_connected_graph, pos, node_size=400, node_color='lightblue')
nx.draw_networkx_edges(fully_connected_graph, pos, edgelist=fully_connected_graph.edges(), width=1.0, alpha=0.5)
nx.draw_networkx_labels(fully_connected_graph, pos, labels, font_size=12, font_color='black')

# Respective costs
edge_labels = {(city1, city2): f"{fully_connected_graph[city1][city2]['weight']:.2f} km" for city1, city2 in fully_connected_graph.edges(
nx.draw_networkx_edge_labels(fully_connected_graph, pos, edge_labels=edge_labels)

# Show plot
plt.title("Fully Connected Graph")
plt.axis('off')
plt.show()


# Total cost
total_cost = sum([mst[city1][city2]['weight'] for city1, city2 in mst.edges])

print(f"Total Cost of Connecting Offices: {total_cost} km")


#OUTPUT
```

```
Minimum Spanning Tree Edges:
Kolkata - Guwahati: 527.1861048687706 km
Kolkata - Delhi: 1308.1441565540097 km
Kolkata - Chennai: 1355.3008742024228 km
Delhi - Udaipur: 567.1391401995337 km
Delhi - Srinagar: 644.0219846952725 km
Time taken to compute MST: 0.0002 seconds
```



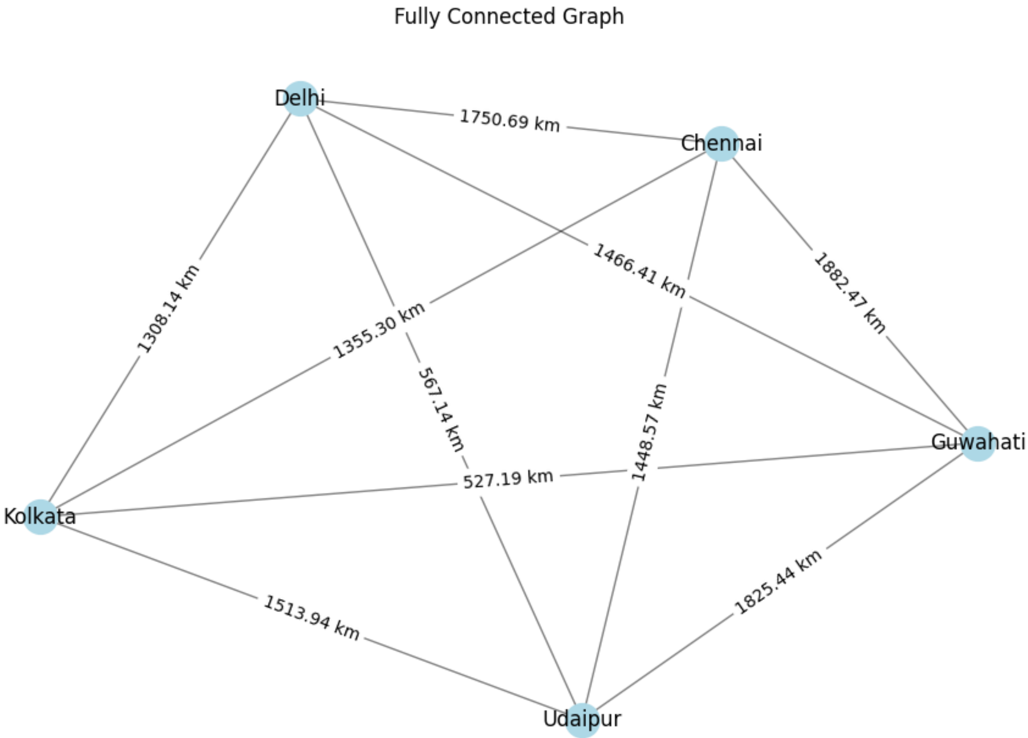

```
Total Cost of Connecting Offices: 4401.792260520009 km
```

5 CITIES

```
from geopy.geocoders import Nominatim
from geopy.distance import geodesic
from itertools import combinations
import networkx as nx
import matplotlib.pyplot as plt
import time

cities = ["Delhi", "Kolkata", "Chennai", "Guwahati", "Udaipur"]

city_coordinates = {}

geolocator = Nominatim(user_agent="telecom_office_connection")

for city in cities:
    location = geolocator.geocode(city)
    if location:
        city_coordinates[city] = (location.latitude, location.longitude)
    else:
        print(f"Location not found for {city}")
edges = []
for city1, city2 in combinations(cities, 2):
    distance = geodesic(city_coordinates[city1], city_coordinates[city2]).kilometers
    edges.append((city1, city2, distance))
edges.sort(key=lambda x: x[2])
mst= nx.Graph()
def find(parent, node):
    if parent[node] == node:
        return node
    return find(parent, parent[node])

parent = {city: city for city in cities}

# Kruskal's Algo
start_time = time.time()

for edge in edges:
    city1, city2, weight = edge
    root1 = find(parent, city1)
    root2 = find(parent, city2)

    if root1 != root2:
        mst.add_edge(city1, city2, weight=weight)
        parent[root1] = root2

end_time = time.time()
execution_time = end_time - start_time

print("Minimum Spanning Tree Edges:")
for edge in mst.edges:
    city1, city2 = edge
    cost = mst[city1][city2]['weight']
    print(f"{city1} - {city2}: {cost} km")
print(f"Time taken to compute MST: {execution_time:.4f} seconds")

import matplotlib.pyplot as plt

#fully connected graph
fully_connected_graph = nx.Graph()

for edge in edges:
    city1, city2, weight = edge
    fully_connected_graph.add_edge(city1, city2, weight=weight)

# Plot
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(fully_connected_graph, seed=42)
labels = {city: city for city in fully_connected_graph.nodes()}

nx.draw_networkx_nodes(fully_connected_graph, pos, node_size=400, node_color='lightblue')
nx.draw_networkx_edges(fully_connected_graph, pos, edgelist=fully_connected_graph.edges(), width=1.0, alpha=0.5)
nx.draw_networkx_labels(fully_connected_graph, pos, labels, font_size=12, font_color='black')

# Respective costs
edge_labels = {(city1, city2): f"{fully_connected_graph[city1][city2]['weight']:.2f} km" for city1, city2 in fully_connected_graph.edges(
nx.draw_networkx_edge_labels(fully_connected_graph, pos, edge_labels=edge_labels)

# Show plot
plt.title("Fully Connected Graph")
```

```
plt.axis('off')
plt.show()


# Total cost
total_cost = sum([mst[city1][city2]['weight'] for city1, city2 in mst.edges])

print(f"Total Cost of Connecting Offices: {total_cost} km")


#OUTPUT
```

```
Minimum Spanning Tree Edges:
Kolkata - Guwahati: 527.1861048687706 km
Kolkata - Delhi: 1308.1441565540097 km
Kolkata - Chennai: 1355.3008742024228 km
Delhi - Udaipur: 567.1391401995337 km
Time taken to compute MST: 0.0002 seconds
```



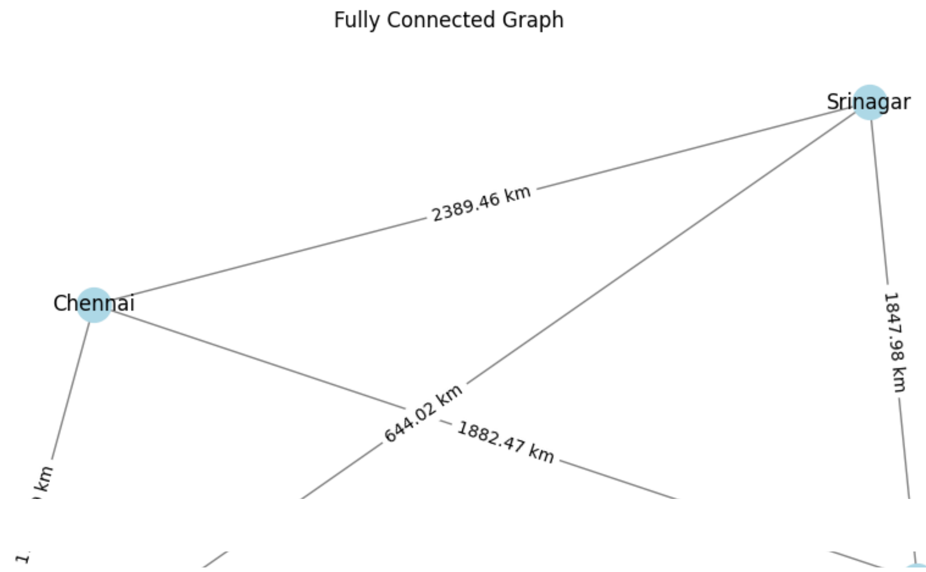

```
Total Cost of Connecting Offices: 3757.770275824737 km
```

4 CITIES

```
from geopy.geocoders import Nominatim
from geopy.distance import geodesic
from itertools import combinations
import networkx as nx
import matplotlib.pyplot as plt
import time

cities = ["Delhi","Chennai", "Guwahati","Srinagar"]

city_coordinates = {}

geolocator = Nominatim(user_agent="telecom_office_connection")

for city in cities:
    location = geolocator.geocode(city)
    if location:
        city_coordinates[city] = (location.latitude, location.longitude)
    else:
        print(f"Location not found for {city}")
edges = []
for city1, city2 in combinations(cities, 2):
    distance = geodesic(city_coordinates[city1], city_coordinates[city2]).kilometers
    edges.append((city1, city2, distance))
edges.sort(key=lambda x: x[2])
```

```
mst= nx.Graph()
def find(parent, node):
    if parent[node] == node:
        return node
    return find(parent, parent[node])

parent = {city: city for city in cities}

# Kruskal's Algo
start_time = time.time()

for edge in edges:
    city1, city2, weight = edge
    root1 = find(parent, city1)
    root2 = find(parent, city2)

    if root1 != root2:
        mst.add_edge(city1, city2, weight=weight)
        parent[root1] = root2

end_time = time.time()
execution_time = end_time - start_time

print("Minimum Spanning Tree Edges:")
for edge in mst.edges:
    city1, city2 = edge
    cost = mst[city1][city2]['weight']
    print(f"{city1} - {city2}: {cost} km")
print(f"Time taken to compute MST: {execution_time:.4f} seconds")

import matplotlib.pyplot as plt

#fully connected graph
fully_connected_graph = nx.Graph()

for edge in edges:
    city1, city2, weight = edge
    fully_connected_graph.add_edge(city1, city2, weight=weight)

# Plot
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(fully_connected_graph, seed=42)
labels = {city: city for city in fully_connected_graph.nodes()}

nx.draw_networkx_nodes(fully_connected_graph, pos, node_size=400, node_color='lightblue')
nx.draw_networkx_edges(fully_connected_graph, pos, edgelist=fully_connected_graph.edges(), width=1.0, alpha=0.5)
nx.draw_networkx_labels(fully_connected_graph, pos, labels, font_size=12, font_color='black')

# Respective costs
edge_labels = {(city1, city2): f"{fully_connected_graph[city1][city2]['weight']:.2f} km" for city1, city2 in fully_connected_graph.edges(
nx.draw_networkx_edge_labels(fully_connected_graph, pos, edge_labels=edge_labels)

# Show plot
plt.title("Fully Connected Graph")
plt.axis('off')
plt.show()


# Total cost
total_cost = sum([mst[city1][city2]['weight'] for city1, city2 in mst.edges])

print(f"Total Cost of Connecting Offices: {total_cost} km")


#OUTPUT
```

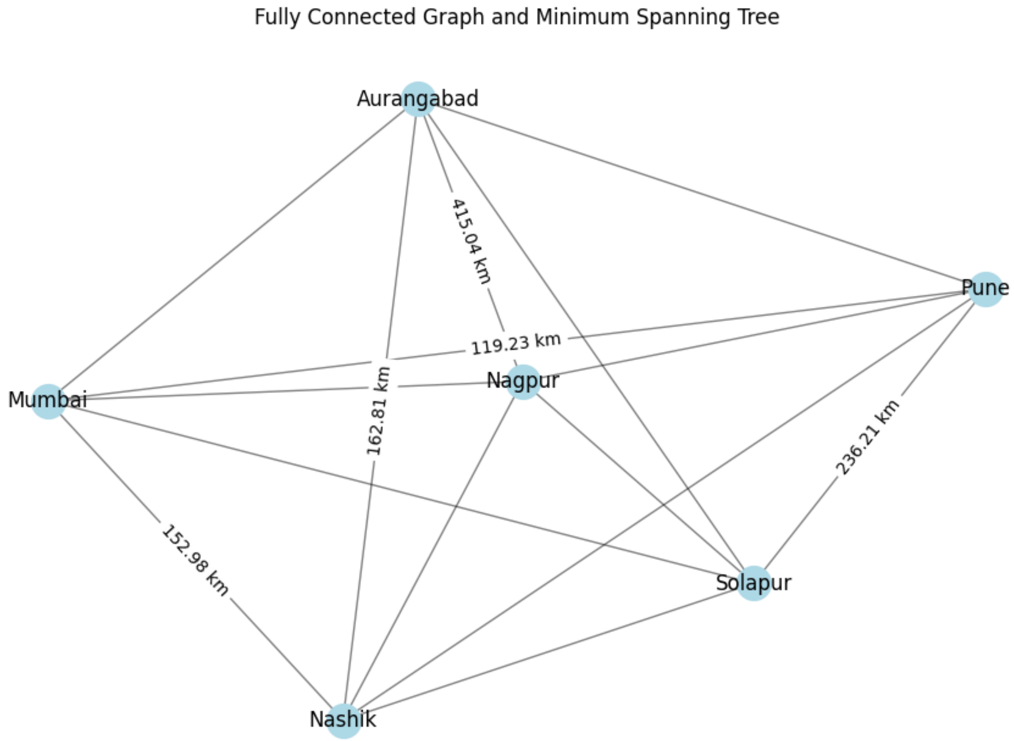
```
Minimum Spanning Tree Edges:
Delhi - Srinagar: 644.0219846952725 km
Delhi - Guwahati: 1466.4073853268937 km
Delhi - Chennai: 1750.6885885785348 km
Time taken to compute MST: 0.0002 seconds
```




```
from geopy.geocoders import Nominatim
from geopy.distance import geodesic
from itertools import combinations
import networkx as nx
import matplotlib.pyplot as plt
import time

cities = ["Mumbai", "Pune", "Nagpur", "Nashik", "Aurangabad", "Solapur"]

city_coordinates = {}

#latitude and longitude for each city
geolocator = Nominatim(user_agent="telecom_office_connection")

for city in cities:
    location = geolocator.geocode(city)
    if location:
        city_coordinates[city] = (location.latitude, location.longitude)
    else:
        print(f"Location not found for {city}")

G = nx.Graph()

# calc distances and add edges to the graph
for city1, city2 in combinations(cities, 2):
    distance = geodesic(city_coordinates[city1], city_coordinates[city2]).kilometers
    G.add_edge(city1, city2, weight=distance)

# MST using Kruskal's algorithm
start_time = time.time()  # Start measuring time

mst = nx.minimum_spanning_tree(G)

end_time = time.time()  # Stop measuring time
execution_time = end_time - start_time

print("Minimum Spanning Tree Edges:")
for edge in mst.edges:
    city1, city2 = edge
    cost = mst[city1][city2]['weight']
    print(f"{city1} - {city2}: {cost} km")

# Print execution time
print(f"Time taken to compute MST: {execution_time:.4f} seconds")

# Plot
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G, seed=42)
labels = {city: city for city in G.nodes()}

nx.draw_networkx_nodes(G, pos, node_size=400, node_color='lightblue')
nx.draw_networkx_edges(G, pos, edgelist=G.edges(), width=1.0, alpha=0.5)
nx.draw_networkx_labels(G, pos, labels, font_size=12, font_color='black')
```

```
# respective costs
edge_labels = {(city1, city2): f"{mst[city1][city2]['weight']:.2f} km" for city1, city2 in mst.edges()}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

# Show plot
plt.title("Fully Connected Graph and Minimum Spanning Tree")
plt.axis('off')
plt.show()

# total cost
total_cost = sum([mst[city1][city2]['weight'] for city1, city2 in mst.edges])

print(f"Total Cost of Connecting Offices: {total_cost} km")
```

```
Minimum Spanning Tree Edges:
Mumbai - Pune: 119.22916693804272 km
Mumbai - Nashik: 152.97575128782637 km
Pune - Solapur: 236.2120988316117 km
Nagpur - Aurangabad: 415.03512074548814 km
Nashik - Aurangabad: 162.80953160402416 km
Time taken to compute MST: 0.0005 seconds
```

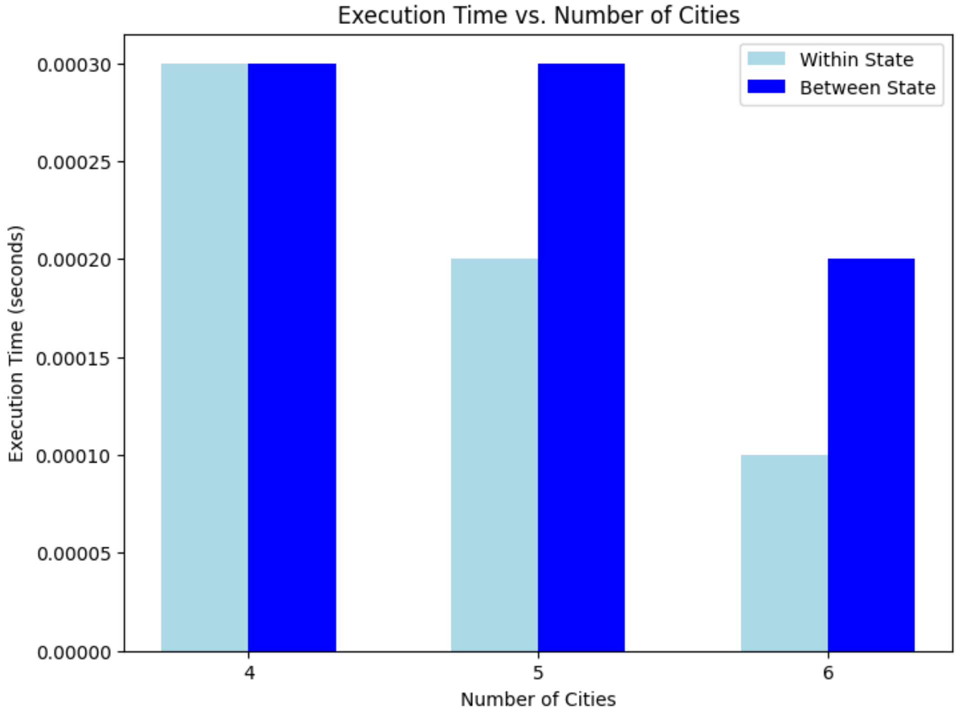



```
Total Cost of Connecting Offices: 1086.2616694069932 km
```

plot graph

```
import matplotlib.pyplot as plt

list_within = [0.0003, 0.0002, 0.0001]  # Within state
list_bet = [0.0003, 0.0003, 0.0002]  # Between state
num_cities = [4, 5, 6]  # Number of cities

fig, ax = plt.subplots(figsize=(8, 6))

bar_width = 0.3

x_within = [x - bar_width/2 for x in num_cities]
x_bet = [x + bar_width/2 for x in num_cities]
bar1 = ax.bar(x_within, list_within, bar_width, label='Within State', color='lightblue')
bar2 = ax.bar(x_bet, list_bet, bar_width, label='Between State', color='b')
ax.set_xlabel('Number of Cities')
ax.set_ylabel('Execution Time (seconds)')
ax.set_title('Execution Time vs. Number of Cities')
ax.set_xticks(num_cities)
ax.set_xticklabels(num_cities)
ax.legend()
```

```
plt.show()
```


Execution Time vs. Number of Cities
Within State
Between State
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Execution Time (seconds)
4
5
6
Number of Cities