In [2]:

```
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp
import pandas as pd
import numpy as np
import folium
import folium.plugins as plugins
import random

# Importing Distance Matrix
df_distance = pd.read_excel('C:\\Users\\ASHIQ\\Desktop\\distance.xlsx', index_col=0)

# Transforming to Numpy Array
distance_matrix = df_distance.to_numpy()

# Creating dictionnary with data
data = {}
data['distance_matrix'] = distance_matrix
print("{:,} destinations".format(len(data['distance_matrix'][0]) - 1))

# Orders quantity (Boxes)
data['demands'] = [0, 9, 11, 8, 12, 10, 7, 13, 11, 9]
# Vehicles Capacities (Boxes)
data['vehicle_capacities'] = [25, 25, 25, 25]

# Fleet informations
# Number of vehicles
data['num_vehicles'] = 4
# Location of the depot
data['depot'] = 0

# Creating Dataframe of Matrix Distance
def create_excel(data):
    n_col = len(data['distance_matrix'][0])
    n_row = len(data['distance_matrix'])
    list_row = ['row' + str(i) for i in range(n_row)]
    list_col = ['col' + str(i) for i in range(n_row)]

    matrix = np.array(data['distance_matrix'])
    df = pd.DataFrame(data=matrix, index=list_row, columns=list_col)
    df.to_excel('df_distance_matrix.xlsx')

# Distance callback
def distance_callback(from_index, to_index):
    """Returns the distance between the two nodes."""
    from_node = manager.IndexToNode(from_index)
    to_node = manager.IndexToNode(to_index)
    return data['distance_matrix'][from_node][to_node]

# Demand callback
def demand_callback(from_index):
    """Returns the demand of the node."""
    from_node = manager.IndexToNode(from_index)
    return data['demands'][from_node]

# Creating the routing index manager.
manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),\
                                       data['num_vehicles'], data['depot'])
```

```
# Creating Routing Model
routing = pywrapcp.RoutingModel(manager)

# Creating and register a transit callback.
transit_callback_index = routing.RegisterTransitCallback(distance_callback)
routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

# Adding Capacity constraint.
demand_callback_index = routing.RegisterUnaryTransitCallback(demand_callback)
routing.AddDimensionWithVehicleCapacity(
    demand_callback_index,
    0,  # null capacity slack
    data['vehicle_capacities'],  # vehicle maximum capacities
    True,  # start cumul to zero
    'Capacity'
)

# Setting first solution heuristic.
search_parameters = pywrapcp.DefaultRoutingSearchParameters()
search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.AUTOMATIC)
search_parameters.local_search_metaheuristic = (
    routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
search_parameters.time_limit.FromSeconds(1)


# Solving the problem.
solution = routing.SolveWithParameters(search_parameters)

if solution:
    total_distance = 0
    total_load = 0
    total_cost = 0  # Total cost variable
    for vehicle_id in range(data['num_vehicles']):
        index = routing.Start(vehicle_id)
        plan_output = 'Route for driver {}:\n'.format(vehicle_id)
        route_distance = 0
        route_load = 0
        route_cost = 0  # Cost for the current route
        while not routing.IsEnd(index):
            node_index = manager.IndexToNode(index)
            route_load += data['demands'][node_index]
            plan_output += ' {0} Parcels({1}) -> '.format(node_index, route_load)
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(previous_index, index, vehicle_id)
            route_cost += route_distance * 0.000497  \
            # Calculate cost for each arc and add to the route cost
        plan_output += ' {0} Parcels({1})\n'.format(manager.IndexToNode(index), route_load)
        plan_output += 'Distance of the route: {} (m)\n'.format(route_distance)
        plan_output += 'Parcels Delivered: {} (parcels)\n'.format(route_load)
        plan_output += 'Cost of the route: ${:.2f}\n'.format(route_cost)\
        # Display cost of the route
        print(plan_output)
        total_distance += route_distance
        total_load += route_load
        total_cost += route_cost  # Accumulate the cost of each route
    print('Total distance of all routes: {:,} (m)'.format(total_distance))
    print('Parcels Delivered: {:,}/{:,}'.format(total_load, sum(data['demands'])))
```

```
    print('Total cost of all routes: ${:.2f}'.format(total_cost))  \
    # Display the total cost of all routes
else:
    print('No Solution')

# Print the optimal solution
print('\nOptimal Solution:')
for vehicle_id in range(data['num_vehicles']):
    index = routing.Start(vehicle_id)
    route = []
    while not routing.IsEnd(index):
        node_index = manager.IndexToNode(index)
        route.append(node_index)
        index = solution.Value(routing.NextVar(index))
    route.append(manager.IndexToNode(index))
    print('Vehicle {}: {}'.format(vehicle_id, route))


# Location Coordinate Values
latitude_values = [24.4583, 24.3194, 24.6417, 24.1746142, 24.3667, 24.45771, \
                   24.2111, 24.4333, 24.5014, 24.6417]
longitude_values = [89.5667, 89.65, 89.5957281, 89.7042, 89.70802, 89.7208, \
                    89.375, 89.5347, 89.65]

# Creating a Folium Map centered at the depot
map_center = (latitude_values[0], longitude_values[0])
m = folium.Map(location=map_center, zoom_start=10)

# Adding depot marker to the map
folium.Marker(location=map_center, tooltip='Depot', icon=folium.Icon(color='black')).add_to(m)

# Generating node coordinates
node_coords = [(lat, lon) for lat, lon in zip(latitude_values, longitude_values)]

# Getting routes for each vehicle
routes = []
route_info = {}
route_colors = ['#'+''.join(random.choices('0123456789ABCDEF', k=6)) \
                for _ in range(data['num_vehicles'])]

for vehicle_id in range(data['num_vehicles']):
    index = routing.Start(vehicle_id)
    route = []
    route_distance = 0
    route_load = 0

    while not routing.IsEnd(index):
        node_index = manager.IndexToNode(index)
        route.append(node_index)
        index = solution.Value(routing.NextVar(index))

    route.append(manager.IndexToNode(index))
    routes.append(route)

    for i in range(len(route) - 1):
        from_node = route[i]
        to_node = route[i + 1]
        route_distance += data['distance_matrix'][from_node][to_node]
        route_load += data['demands'][to_node]
```

```
    route_info[vehicle_id] = {'distance': route_distance, 'load': route_load}

# Adding node markers and route polylines to the map
route_markers = folium.FeatureGroup(name="Route Markers")
route_lines = folium.FeatureGroup(name="Route Lines")

for vehicle_id, route in enumerate(routes):
    # Geting coordinates for the route
    route_coords = [node_coords[node_id - 1] \
                    for node_id in route if node_id - 1 < len(node_coords)]

    # Getting color for the route
    color = route_colors[vehicle_id]

    # Adding markers for the route nodes
    for node_id, coords in zip(route, route_coords):
        marker = folium.Marker(location=coords, tooltip=f"Node {node_id}",\
                               icon=folium.Icon(color=color))
        marker.add_to(route_markers)

    # Adding polyline to the route lines with the assigned color
    line = folium.PolyLine(locations=route_coords, color=color, weight=2.5, \
                           opacity=1, popup=f"Distance: {route_info[vehicle_id]['distance']} \
                           (m)\nLoad: {route_info[vehicle_id]['load']} (parcels)")
    line.add_to(route_lines)

# Adding cluster markers if multiple nodes have the same coordinates
cluster_markers = plugins.MarkerCluster().add_to(m)
for coords, color in zip(node_coords, route_colors):
    folium.Marker(location=coords, icon=folium.Icon(color=color)).add_to(cluster_markers)

# Adding route markers and lines to the map
route_markers.add_to(m)
route_lines.add_to(m)

# Adding layer control to the map
folium.LayerControl().add_to(m)

# Displaying the map
m
```