Date:22.10.25

TASK:11

VEHICLE ROUTING PROBLEM.

PROBLEM STATEMENT :

In logistics and distribution, efficient routing of vehicles to service a set of customers is critical. The problem is to determine the optimal set of routes for a fleet of vehicles starting from a depot to deliver goods to multiple customers with known demands, such that the total route cost or distance is minimized and constraints like vehicle capacity and delivery requirements are satisfied.

AIM: To implement an optimized solution for the Vehicle Routing Problem (VRP) that minimizes the total cost or distance of delivery routes while meeting all constraints such as vehicle capacity and demand.

OBJECTIVE:

1. To understand the concept and real-world application of the VRP.
2. To develop a program that generates optimal routes for a fleet of delivery vehicles.
3. To minimize the total cost, distance, or time in vehicle routing.
4. To ensure each customer is visited exactly once by a single vehicle.
5. To adhere to vehicle capacity and delivery constraints.
6. To visualize or print the optimized routes.

DESCRIPTION :

Vehicle Routing Problem (VRP) is a well-known combinatorial optimization problem. It involves determining the most efficient routes for multiple vehicles to deliver goods to a given set of customers. The classical VRP assumes:

- A central depot where vehicles start and end their routes.
- A fleet of identical vehicles with a maximum capacity.
- A number of customers with known locations and demand.

The goal is to service all customers while minimizing the total distance traveled, number of vehicles used, or total delivery cost.

ALGORITHM:

Step 1:Input the data:

a.Distance matrix between each pair of locations (depot + customers).
b.Number of vehicles.
c.Vehicle capacity.
d.Customer demands.

Step 2:Initialize the Routing Manager and Model:
a.Create nodes representing customers and depot.
b.Set vehicle start and end at the depot.

Step 3: Define the distance and demand callbacks:
a.These functions return distances and demands for use in constraints.

Step 4: Set the objective function:
a.Minimize the total travel distance.

Step 5:Apply capacity constraints:
a.Ensure no vehicle exceeds its capacity.

Step 6:Solve the routing problem:
a.Find optimal or near-optimal routes.

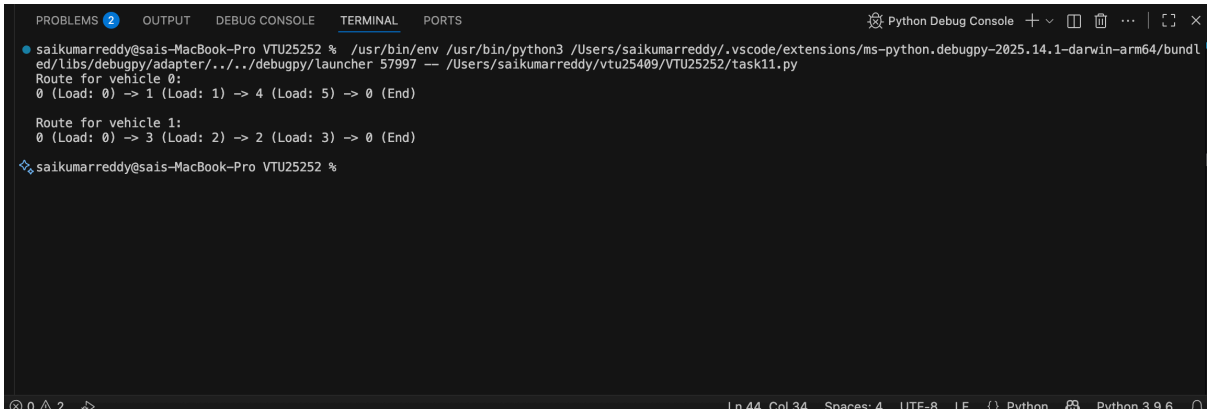
PROGRAM :

```
from ortools.constraint_solver import pywrapcp, routing_enums_pb2

def create_data_model():
    data = {}
    data['distance_matrix'] = [
        [0, 9, 7, 14, 18],
        [9, 0, 10, 15, 11],
        [7, 10, 0, 8, 9],
        [14, 15, 8, 0, 6],
        [18, 11, 9, 6, 0],
    ]
    data['demands'] = [0, 1, 1, 2, 4]  # 0 is depot
    data['vehicle_capacities'] = [5, 5]
    data['num_vehicles'] = 2
    data['depot'] = 0
    return data

def main():
    data = create_data_model()

    manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),
                                           data['num_vehicles'],
                                           data['depot'])

    routing = pywrapcp.RoutingModel(manager)
```

```
    def distance_callback(from_index, to_index):
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return data['distance_matrix'][from_node][to_node]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback)
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    def demand_callback(from_index):
        return data['demands'][manager.IndexToNode(from_index)]

    demand_callback_index =
routing.RegisterUnaryTransitCallback(demand_callback)
    routing.AddDimensionWithVehicleCapacity(
        demand_callback_index,
        0,
        data['vehicle_capacities'],
        True,
        'Capacity'
    )

    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = (
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)

    solution = routing.SolveWithParameters(search_parameters)

    if solution:
        for vehicle_id in range(data['num_vehicles']):
            index = routing.Start(vehicle_id)
            route = 'Route for vehicle {}:\n'.format(vehicle_id)
            route_load = 0
            while not routing.IsEnd(index):
                node_index = manager.IndexToNode(index)
                route_load += data['demands'][node_index]
                route += f'{node_index} (Load: {route_load}) -> '
                index = solution.Value(routing.NextVar(index))
            node_index = manager.IndexToNode(index)
            route += f'{node_index} (End)\n'
            print(route)
    else:
        print("No solution found.")

if __name__ == '__main__':
```

main()

OUTPUT :


PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python Debug Console
saikumarreddy@sais-MacBook-Pro VTU25252 % /usr/bin/env /usr/bin/python3 /Users/saikumarreddy/.vscode/extensions/ms-python.debugpy-2025.14.1-darwin-arm64/bundl
ed/libs/debugpy/adapter/../../debugpy/launcher 57997 -- /Users/saikumarreddy/vtu25409/VTU25252/task11.py
Route for vehicle 0:
0 (Load: 0) -> 1 (Load: 1) -> 4 (Load: 5) -> 0 (End)
Route for vehicle 1:
0 (Load: 0) -> 3 (Load: 2) -> 2 (Load: 3) -> 0 (End)
saikumarreddy@sais-MacBook-Pro VTU25252 %

RESULT: This project successfully implements a basic version of the Vehicle Routing Problem using Python and Google OR-Tools