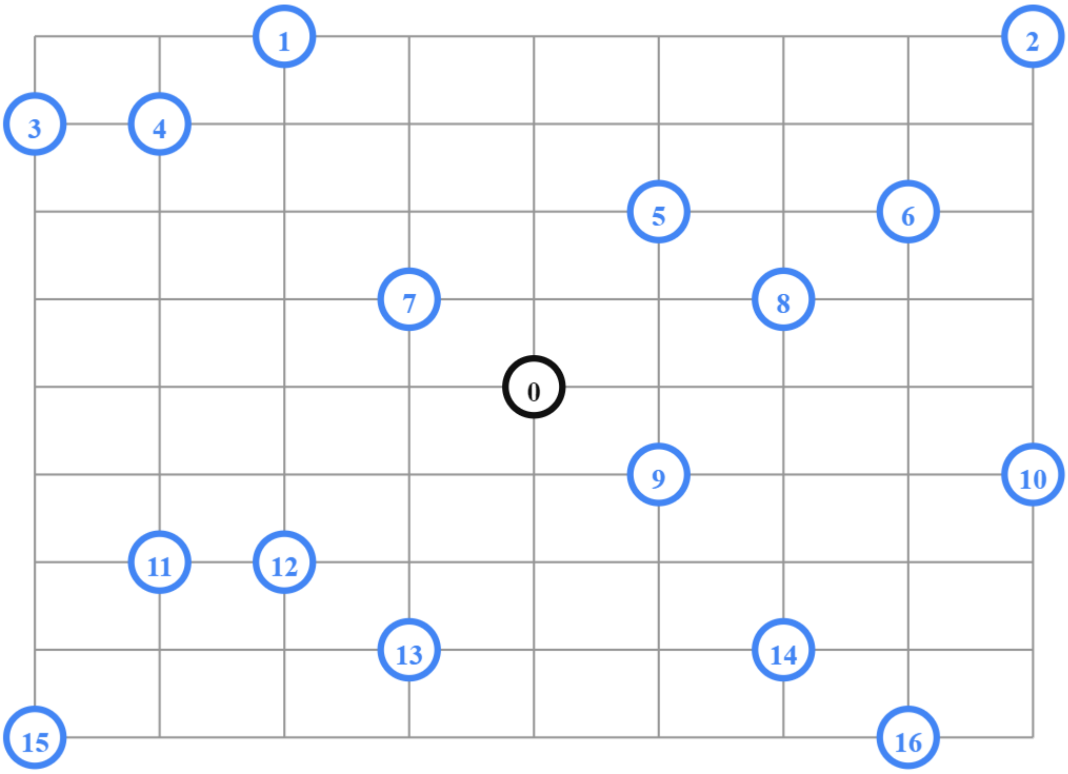
VRP Code

The goal is to find optimal routes for multiple vehicles visiting a set of locations. (When there's only one vehicle, it reduces to the Traveling Salesman Problem.)

optimal routes -> minimize the length of the longest single route among all vehicles.



Creating The Data: -

For this problem, we take node 0 as the depot and assume number of vehicles = 4

```
def create_data_model():
    # Stores the data for the problem.
    data = {}
    data['distance_matrix'] = [
        [
            0, 548, 776, 696, 582, 274, 502, 194, 308, 194, 536, 502, 388, 
354,
            468, 776, 662
        ],
        [
            548, 0, 684, 308, 194, 502, 730, 354, 696, 742, 1084, 594, 480, 
674,
            1016, 868, 1210
        ],
        [
            776, 684, 0, 992, 878, 502, 274, 810, 468, 742, 400, 1278, 
1164,
            1130, 788, 1552, 754
        ],
        [
            696, 308, 992, 0, 114, 650, 878, 502, 844, 890, 1232, 514, 628, 
822,
```

```
            1164, 560, 1358
        ],
        [
            582, 194, 878, 114, 0, 536, 764, 388, 730, 776, 1118, 400, 514,
708,
            1050, 674, 1244
        ],
        [
            274, 502, 502, 650, 536, 0, 228, 308, 194, 240, 582, 776, 662,
628,
            514, 1050, 708
        ],
        [
            502, 730, 274, 878, 764, 228, 0, 536, 194, 468, 354, 1004, 890,
856,
            514, 1278, 480
        ],
        [
            194, 354, 810, 502, 388, 308, 536, 0, 342, 388, 730, 468, 354,
320,
            662, 742, 856
        ],
        [
            308, 696, 468, 844, 730, 194, 194, 342, 0, 274, 388, 810, 696,
662,
            320, 1084, 514
        ],
        [
            194, 742, 742, 890, 776, 240, 468, 388, 274, 0, 342, 536, 422,
388,
            274, 810, 468
        ],
        [
            536, 1084, 400, 1232, 1118, 582, 354, 730, 388, 342, 0, 878,
764,
            730, 388, 1152, 354
        ],
        [
            502, 594, 1278, 514, 400, 776, 1004, 468, 810, 536, 878, 0,
114,
            308, 650, 274, 844
        ],
        [
            388, 480, 1164, 628, 514, 662, 890, 354, 696, 422, 764, 114, 0,
194,
            536, 388, 730
        ],
        [
            354, 674, 1130, 822, 708, 628, 856, 320, 662, 388, 730, 308,
194, 0,
            342, 422, 536
        ],
        [
            468, 1016, 788, 1164, 1050, 514, 514, 662, 320, 274, 388, 650,
536,
            342, 0, 764, 194
        ],
        [
            776, 868, 1552, 560, 674, 1050, 1278, 742, 1084, 810, 1152,
274,
            388, 422, 764, 0, 798
```

```
        ],
        [
            662, 1210, 754, 1358, 1244, 708, 480, 856, 514, 468, 354, 844,
730,
            536, 194, 798, 0
        ],
    ]
    data['num_vehicles'] = 4
    data['depot'] = 0
    return data
```

To set up the example and compute the distance matrix, assign the following *x-y* coordinates to the locations shown in the city diagram (This is NOT needed for solving the VRP, but is used for better understanding. The Manhattan Distance method has been used to get these values):

```
[(456, 320), # location 0 - the depot
(228, 0),    # location 1
(912, 0),    # location 2
(0, 80),     # location 3
(114, 80),   # location 4
(570, 160),  # location 5
(798, 160),  # location 6
(342, 240),  # location 7
(684, 240),  # location 8
(570, 400),  # location 9
(912, 400),  # location 10
(114, 480),  # location 11
(228, 480),  # location 12
(342, 560),  # location 13
(684, 560),  # location 14
(0, 640),    # location 15
(798, 640)]  # location 16
```

Like in TSP, we need to create a 'distance call-back function' which returns the distance between any 2 nodes.

```
def distance_callback(from_index, to_index):
    # Convert from routing variable Index to distance matrix NodeIndex.
    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)
```

We use the dynamic programming approach. So, we compute the cumulative distance travelled by each vehicle along its route.

To create the distance dimension, we use the 'AddDimension' method. The argument 'transit_callback_index' is the index for the distance_callback.

```
dimension_name = 'Distance'
routing.AddDimension(
    transit_callback_index,
    0,  # no slack
    3000,  # vehicle maximum travel distance
    True,  # start cumul to zero
    dimension_name)
distance_dimension = routing.GetDimensionOrDie(dimension_name)
distance_dimension.SetGlobalSpanCostCoefficient(100)
```

#The method SetGlobalSpanCostCoefficient sets a large coefficient (100) for the global span of the routes. Here, it is the maximum of the distances of the routes.

Solution function: -

```
def print_solution(data, manager, routing, solution):
    #Prints solution on console.
    max_route_distance = 0
    for vehicle_id in range(data['num_vehicles']):
        index = routing.Start(vehicle_id)
        plan_output = 'Route for vehicle {}:\n'.format(vehicle_id)
        route_distance = 0
        while not routing.IsEnd(index):
            plan_output += ' {} -> '.format(manager.IndexToNode(index))
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(
                previous_index, index, vehicle_id)
        plan_output += '{}\n'.format(manager.IndexToNode(index))
        plan_output += 'Distance of the route: 
{}m\n'.format(route_distance)
        print(plan_output)
        max_route_distance = max(route_distance, max_route_distance)
    print('Maximum of the route distances: {}m'.format(max_route_distance))
```

‘main’ function: -

```
def main():
    # Instantiate the data problem.
    data = create_data_model()

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

    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)


    # Create and register a transit callback.
```

```
    def distance_callback(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        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)

    # Define cost of each arc.
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Add Distance constraint.
    dimension_name = 'Distance'
    routing.AddDimension(
        transit_callback_index,
        0,  # no slack
        3000,  # vehicle maximum travel distance
        True,  # start cumul to zero
        dimension_name)
    distance_dimension = routing.GetDimensionOrDie(dimension_name)
    distance_dimension.SetGlobalSpanCostCoefficient(100)

    # Setting first solution heuristic.
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = (
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)

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

    # Print solution on console.
    if solution:
        print_solution(data, manager, routing, solution)


if __name__ == '__main__':
    main()
```

Output: - (Screenshot)

```
Route for vehicle 0:
 0 -> 8 -> 6 -> 2 -> 5 -> 0
Distance of route: 1552m

Route for vehicle 1:
 0 -> 7 -> 1 -> 4 -> 3 -> 0
Distance of route: 1552m

Route for vehicle 2:
 0 -> 9 -> 10 -> 16 -> 14 -> 0
Distance of route: 1552m

Route for vehicle 3:
 0 -> 12 -> 11 -> 15 -> 13 -> 0
Distance of route: 1552m

Total distance of all routes: 6208m
```