Name : Akash Kulkarni

Class : TE-10

Batch : K-10

Roll No : 33241

Assn : 1

Problem Statement :

Write a program to implement Travelling Salesperson Problem using appropriate heuristic and search strategy.

In [1]:

```python
import copy
```

In [2]:

```python
inf = float('inf')
```

In [3]:

```python
class TSP_AI:
    """Traveling Salesman Problem
        -------------------------------------------------------------------
        Traveling Salesman Problem using Nearest Neighbour AI algorithm
    """

    def __init__(self, city_matrix = None, source = 0):
        self.city_matrix = [[0]*4]*4 if city_matrix is None else city_matrix
        self.n : int = len(self.city_matrix)
        self.source : int = source


    def Input(self):
        self.n = int(input('Enter city count : '))

        for i in range(self.n):                                      # Get the distances between cities
            self.city_matrix.append([
                inf if i == j else int(input(f'Cost to travel from city {i+1} to {j+1} : '))
                for j in range(self.n)
            ])

        self.source = int(input('Source: ')) % self.n                # Get the source city


    def solve(self):
        minCost = inf                                                # Initially minCost is infinity
        for i in range(self.n):
            print("Path", end='')
            cost = self._solve(copy.deepcopy(city_matrix), i, i)         # Calling solver for each as source cit
            print(f" -> {i+1}     :    Cost = {cost}")
            if cost and cost < minCost: minCost = cost               # If this cost is optimal, save it

        return minCost


    def _solve(self, city_matrix, currCity = 0, source = 0):
        if self.n < 2: return 0
        print(f" -> {currCity+1}", end='')

        for i in range(self.n):
            city_matrix[i][currCity] = inf                           # Set all values in the currCity colum

        currMin, currMinPos = inf, 0
        for j in range(self.n):
            if currMin > city_matrix[currCity][j]:                   # Get the nearest city to the current c
                currMin, currMinPos = city_matrix[currCity][j], j

        if currMin == inf: return self.city_matrix[currCity][source]     # If currMin is infinity(i.e. all citie
        city_matrix[currCity][currMinPos] = city_matrix[currMinPos][currCity] = inf      # Set distance from cu
        return currMin + self._solve(city_matrix, currMinPos, source)    # Calling the next recursion for select
```

In [4]:

```python
if __name__ == '__main__':
    city_matrix = [
        [inf, 10,  15,  20],
        [10,  inf, 35,  25],
        [15,  35,  inf, 30],
        [20,  25,  30,  inf]
    ]

    source_city = 0
    tsp = TSP_AI(city_matrix, source_city)
    print(f"Optimal Cost : {tsp.solve()}")
```

```
Path -> 1 -> 2 -> 4 -> 3 -> 1     :    Cost = 80
Path -> 2 -> 1 -> 3 -> 4 -> 2     :    Cost = 80
Path -> 3 -> 1 -> 2 -> 4 -> 3     :    Cost = 80
Path -> 4 -> 1 -> 2 -> 3 -> 4     :    Cost = 95
Optimal Cost : 80
```

In [ ]: