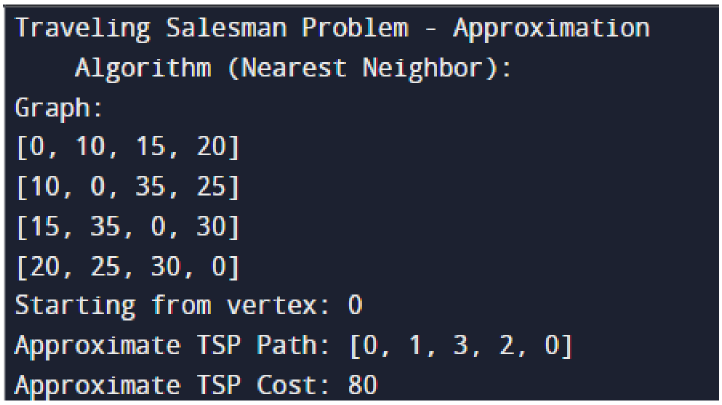
123. NP-Complete and NP-Hard Problem

AIM: To solve the NP – Complete and NP – Hard Problem by using Tracibility and approximation Algorithm

PROGRAM:

```
class TSP:
    def __init__(self, graph):
        self.graph = graph
        self.num_vertices = len(graph)

    def nearest_neighbor(self, start_vertex):
        visited = [False] * self.num_vertices
        path = [start_vertex]
        current_vertex = start_vertex
        total_cost = 0

        visited[start_vertex] = True

        for _ in range(self.num_vertices - 1):
            next_vertex = None
            min_cost = float('inf')  # Initialize min_cost to infinity

            for neighbor in range(self.num_vertices):
                if not visited[neighbor] and self.graph[current_vertex][neighbor] < min_cost:
                    min_cost = self.graph[current_vertex][neighbor]
                    next_vertex = neighbor

            path.append(next_vertex)
            total_cost += min_cost
            visited[next_vertex] = True
            current_vertex = next_vertex

        # Return to the start vertex
        path.append(start_vertex)
        total_cost += self.graph[current_vertex][start_vertex]

        return path, total_cost

# Example usage:
graph = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]

tsp = TSP(graph)
start_vertex = 0  # Start from vertex 0
path, total_cost = tsp.nearest_neighbor(start_vertex)

print("Traveling Salesman Problem - Approximation Algorithm (Nearest Neighbor):")
print("Graph:")
for row in graph:
    print(row)
print("Starting from vertex:", start_vertex)
print("Approximate TSP Path:", path)

print("Approximate TSP Cost:", total_cost)
```

OUTPUT: 

```
Traveling Salesman Problem - Approximation
    Algorithm (Nearest Neighbor):
Graph:
[0, 10, 15, 20]
[10, 0, 35, 25]
[15, 35, 0, 30]
[20, 25, 30, 0]
Starting from vertex: 0
Approximate TSP Path: [0, 1, 3, 2, 0]
Approximate TSP Cost: 80
```

TIME COMPLEXITY: O(N log N)