	Bansilal Ramnath Agarwal Charitable Trust's Vishwakarma Institute of Information Technology **Department of** **Artificial Intelligence and Data Science**

Student Name: **Sahil Dilip Sonawane**		
Class: **TY**	Division: **C**	Roll No.: **373057**
Semester: **5**	Academic Year: **2023-24**	
Subject Name & Code: **Design and Analysis of Algorithm: ADUA31202**		
Title of Assignment: **To implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using different algorithmic strategies and determine the optimal solution.**		
Date of Performance:	Date of Submission:	

Assignment No. 8

Aim:

To implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using different algorithmic strategies and determine the optimal solution.

Problem Statement:

Implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using different algorithmic strategies and determine the optimal solution.

Software Requirements:

Text Editor: VSCode, Online GDB Compiler
Environment: Python

Theory:

Travelling Sales Person Problem-

The traveling salesman problems abide by a salesman and a set of cities. The salesman has to visit every one of the cities starting from a certain one (e.g., the hometown) and to return to the same city. The challenge of the problem is that the traveling salesman needs to minimize the total length of the trip.

Suppose the cities are x1 x2..... xn where cost cij denotes the cost of travelling from city xi to xj. The travelling salesperson problem is to find a route starting and ending at x1 that will take in all cities with the minimum cost.

Using Dynamic Programming Approach:

Let the given set of vertices be {1, 2, 3, 4,….n}. Let us consider 1 as starting and ending point of output. For every other vertex I (other than 1), we find the minimum cost path with 1 as the starting point, I as the ending point, and all vertices appearing exactly once. Let the cost of this path cost (i), and the cost of the corresponding Cycle would cost (i) + dist(i, 1) where dist(i, 1) is the distance from I to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This looks simple so far.

Let us define a term C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i. We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.

If size of S is 2, then S must be {1, i},
C(S, i) = dist(1, i)
Else if size of S is greater than 2.
C(S, i) = min { C(S-{i}, j) + dist(j, i)} where j belongs to S, j != i and j != 1.

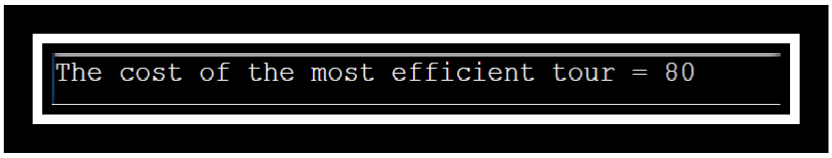
Algorithm:

If size of S is 2, then S must be {1, i},
C(S, i) = dist(1, i)
Else if size of S is greater than 2.
C(S, i) = min { C(S-{i}, j) + dist(j, i)} where j belongs to S, j != i and j != 1.

Program Code:

```
n = 4
MAX = 1000000
dist = [
  [0, 0, 0, 0, 0],
  [0, 0, 10, 15, 20],
  [0, 10, 0, 25, 25],
  [0, 15, 25, 0, 30],
  [0, 20, 25, 30, 0],
]

memo = [[0] * (1 << (n + 1)) for _ in range(n + 1)]

def fun(i, mask):
  if mask == ((1 << i) | 3):
    return dist[1][i]
  if memo[i][mask] != 0:
    return memo[i][mask]

  res = MAX

  for j in range(1, n + 1):
    if (mask & (1 << j)) and j != i and j != 1:
      res = min(res, fun(j, mask & (~(1 << i))) + dist[j][i])

  memo[i][mask] = res
  return res

def main():
  ans = MAX
  for i in range(1, n + 1):
    ans = min(ans, fun(i, (1 << (n + 1)) - 1) + dist[i][1])

  print("The cost of the most efficient tour =", ans)

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

Output:



```
The cost of the most efficient tour = 80
```

Using Back Tracking Approach:

Algorithm:

Consider city 1 (let say 0th node) as the starting and ending point. Since route is cyclic, we can consider any point as starting point.

Start traversing from the source to its adjacent nodes in dfs manner.

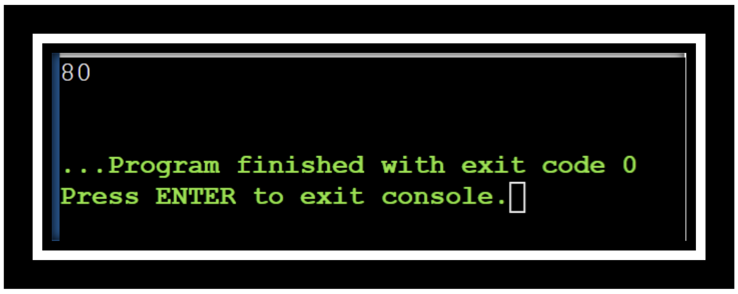
Calculate cost of every traversal and keep track of minimum cost and keep on updating the value of minimum cost stored value.

Return the permutation with minimum cost.

Program Code:

```
V = 4

def tsp(graph, v, curr_pos, n, count, cost, ans):
   if count == n and graph[curr_pos][0]:
      ans[0] = min(ans[0], cost + graph[curr_pos][0])
      return

   for i in range(n):
      if not v[i] and graph[curr_pos][i]:
         v[i] = True
         tsp(graph, v, i, n, count + 1, cost + graph[curr_pos][i], ans)
         v[i] = False

if __name__ == "__main__":
   n = 4

   graph = [
      [0, 10, 15, 20],
      [10, 0, 35, 25],
      [15, 35, 0, 30],
      [20, 25, 30, 0]
   ]

   v = [False] * n
   v[0] = True

   ans = [float('inf')]
   tsp(graph, v, 0, n, 1, 0, ans)
   print(ans[0])
```

Output:



```
80


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

In this assignment we successfully found the optimal solution for the Traveling Salesperson problem. We then solved the same problem instance using Dynamic Programming Approach and Backtracking Approach and determined the optimal solution.
