**PROGRAM:**

import random

import copy

# Calculate total path cost

def calculate\_cost(graph, path):

cost = 0

for i in range(len(path) - 1):

cost += graph[path[i]][path[i + 1]]

cost += graph[path[-1]][path[0]] # Return to start

return cost

# Generate neighbors by swapping two cities

def get\_neighbors(path):

neighbors = []

for i in range(1, len(path)):

for j in range(i + 1, len(path)):

neighbor = path.copy()

neighbor[i], neighbor[j] = neighbor[j], neighbor[i]

neighbors.append(neighbor)

return neighbors

# Hill Climbing Algorithm

def hill\_climbing\_tsp(graph):

V = len(graph)

current\_path = list(range(V)) # Initial path: [0, 1, 2, ..., V-1]

random.shuffle(current\_path) # Randomize the initial path

current\_cost = calculate\_cost(graph, current\_path)

while True:

neighbors = get\_neighbors(current\_path)

next\_path = current\_path

next\_cost = current\_cost

for neighbor in neighbors:

cost = calculate\_cost(graph, neighbor)

if cost < next\_cost:

next\_path = neighbor

next\_cost = cost

# If no better neighbor, stop (local minimum)

if next\_cost >= current\_cost:

break

current\_path = next\_path

current\_cost = next\_cost

return current\_path, current\_cost

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

graph = [[0, 10, 15, 20],

[10, 0, 35, 25],

[15, 35, 0, 30],

[20, 25, 30, 0]]

best\_path, best\_cost = hill\_climbing\_tsp(graph)

print("Best path found:", best\_path)

print("Cost of path:", best\_cost)

**OUTPUT:**

