LAB-4

Aim: Program to implement the concept of hill climbing.

Algorithm:

Evaluate the initial state.

Loop until a solution is found or there are no new operators left to be applied:

- Select and apply a new operator
- Evaluate the new state:

  goal -⟶ quit

  better than current state -⟶ new current state

Source Code:

```
import random
# Distance matrix representing distances between cities
# Replace this with the actual distance matrix for your problem
distance_matrix = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]
def total_distance(path):
    # Calculate the total distance traveled in the given path
    total = 0
    for i in range(len(path) - 1):
        total += distance_matrix[path[i]][path[i+1]]
    total += distance_matrix[path[-1]][path[0]]  # Return to starting city
    return total
```

```
def hill_climbing_tsp(num_cities, max_iterations=10000):
    current_path = list(range(num_cities))  # Initial solution, visiting cities in order
    current_distance = total_distance(current_path)
    for _ in range(max_iterations):
        # Generate a neighboring solution by swapping two random cities
        neighbor_path = current_path.copy()
        i, j = random.sample(range(num_cities), 2)
        neighbor_path[i], neighbor_path[j] = neighbor_path[j], neighbor_path[i]
        neighbor_distance = total_distance(neighbor_path)

        # If the neighbor solution is better, move to it
        if neighbor_distance < current_distance:
            current_path = neighbor_path
            current_distance = neighbor_distance
    return current_path
def main():
    num_cities = 4  # Number of cities in the TSP
    solution = hill_climbing_tsp(num_cities)
    print("Optimal path:", solution)
    print("Total distance:", total_distance(solution))
if __name__ == "__main__":
    main()
```

OUTPUT:

Optimal path: [1, 0, 2, 3]
Total distance: 80