**Date:24.10.25**

**TASK:11**

**ROBOT TRAVERSAL**

**CO1, CO2, CO3 S3**

**PROBLEM STATEMENT :**

A robot is placed on a 2D grid of size M × N (M rows and N columns). The robot starts from a specific cell (usually the top-left corner (0,0)) and needs to reach the bottom-right corner (M−1, N−1). The robot can move only in specific directions — typically right or down

**AIM :**

To develop an algorithm that enables a robot to move from the starting position to the destination position in a grid by finding a valid path while avoiding obstacles.

**OBJECTIVE :**

1. Define a state‐space model for weather variables (rainfall, temperature, humidity, wind) across discrete time steps (e.g., next 24 h).
2. Implement a heuristic search algorithm (e.g., A\* or greedy + beam search) over the state-space to predict likely next states.
3. Use an evolutionary algorithm (e.g., Genetic Algorithm) to optimise forecasting model hyper-parameters or transition heuristics.
4. Formulate planning or constraint‐satisfaction problem (CSP) to enforce constraints such as temporal continuity, spatial adjacency, and physical limits (e.g., humidity + temperature bounds).
5. Evaluate forecast accuracy (e.g., RMSE, MAE) and constraint adherence.

**DESCRIPTION :**

1. **State‐space representation**: Each state at time t is described by a vector of weather variables for a given region (or discretised cells).
2. **Transitions**: From a state at time t, possible successor states at time t + Δt can be generated via a model (e.g., learned or rule-based) + search.
3. **Heuristic search**: Use a heuristic that estimates “distance” (error) to observed climatological or extreme event state. For example, lower heuristic cost if humidity and temperature align with typical rainfall threshold.
4. **Evolutionary optimisation**: Evolve a population of candidate models/heuristics (or parameters of transition functions) to minimize forecast error.
5. **Planning/CSP**: Frame constraints such as “if rainfall > X then humidity must be > Y”, “wind speed change between consecutive hours must ≤ Z”, or “spatially adjacent cells cannot differ beyond a threshold X”. Solve using CSP solver to pick a consistent forecast sequence.
6. **Integration**: The module outputs a forecast of rainfall (and perhaps temperature/humidity) for the next 24 h at hourly (or three-hour) intervals, for the region.

**ALGORITHM :**

* **Load historical weather data for the chosen region and variables (rainfall, temperature, humidity, wind).**
* **Pre-process data: handle missing values, normalise or discretise variables, possibly map into spatial grid/time steps.**
* **Define the *state space* representation: each state corresponds to weather variables at a particular time (and possibly location).**
* **Define the *transition model*: how one state evolves into the next (could be based on learned model, rules, heuristics).**
* **Define a *heuristic function* that estimates a cost or “distance” from a desired/target state (e.g., rainfall > threshold, extreme event).**
* **Run a heuristic search (e.g., A\* or greedy) through the state-space**

**PROGRAM :**

**# Robot Traversal Problem in Python**

**def is\_safe(x, y, grid, visited):**

**"""Check if the next move is within the grid and on a free cell."""**

**rows = len(grid)**

**cols = len(grid[0])**

**return (0 <= x < rows) and (0 <= y < cols) and (grid[x][y] == 0) and (not visited[x][y])**

**def find\_paths(grid, x, y, path, all\_paths, visited):**

**"""Recursive function to find all possible paths."""**

**rows = len(grid)**

**cols = len(grid[0])**

**grid = [**

**[0, 0, 0],**

**[0, 1, 0],**

**[0, 0, 0]**

**]**

**visited = [[False for \_ in range(len(grid[0]))] for \_ in range(len(grid))]**

**all\_paths = []**

**find\_paths(grid, 0, 0, [], all\_paths, visited)**

**# Display Output**

**if all\_paths:**

**print("All possible paths from start to destination:")**

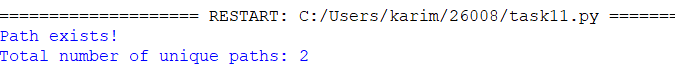
**for i, p in enumerate(all\_paths, 1):**

**print(f"Path {i}: {' -> '.join(p)}")**

**else:**

**print("No path exists!")**

**OUTPUT :**

****

**CONCLUSION :**

The **Robot Traversal problem** demonstrates how a robot can intelligently navigate a grid-based environment by finding valid paths from a **starting point** to a **destination** while avoiding obstacles.