Experiment No. 8

Aim:

To write a program to implement the **Hill Climbing Algorithm**.

Theory:

Hill Climbing is a **heuristic search algorithm** used for mathematical optimization problems in Artificial Intelligence. It is an **iterative algorithm** that continuously moves in the direction of increasing (or decreasing) value — in other words, toward the "top of the hill" (maximum) or "bottom of the valley" (minimum) depending on the problem.

The algorithm starts with an arbitrary solution to a problem and iteratively makes small changes to the solution. If the change produces a better solution, the new solution becomes the current one. The process continues until no further improvement can be found.

Key Concepts:

- **Heuristic Function (h(n))**:
  Evaluates how good a given state is relative to the goal.
- **Current State**:
  The present node or solution in the search space.
- **Neighboring States**:
  Possible next states that can be reached by a small change to the current state.
- **Termination Condition**:
  The algorithm stops when it reaches a state where no neighboring state has a better heuristic value.

Types of Hill Climbing:

1. **Simple Hill Climbing:**
   Selects the first neighbor that improves the current state.
2. **Steepest-Ascent Hill Climbing:**
   Examines all neighbors and selects the one with the best improvement.
3. **Stochastic Hill Climbing:**
   Selects a random neighbor and decides probabilistically whether to move to it.

Limitations:

- May get stuck in **local maxima**.

- May stop at a **plateau** (flat area of equal heuristic values).
- May loop in a **ridge** area.

Time and Space Complexity:

- **Time Complexity:** $O(b^d)$, where b = branching factor and d = depth of search.
- **Space Complexity:** $O(b^d)$, as it stores the search path.

Algorithm:

Hill Climbing Algorithm Steps:

1. Start with an initial random solution (current state).
2. Evaluate the heuristic value of the current state.
3. Generate neighboring states (possible moves).
4. Select the neighbor with the best heuristic value.
5. If the neighbor's value is better than the current state's, move to that neighbor.
6. Repeat steps 2–5 until no better neighbors exist (i.e., a peak is reached).
7. Return the current state as the optimal solution found.

Code (Java):

```
// Java program to implement Hill Climbing Algorithm
// Experiment 8 - Artificial Intelligence Practical

import java.util.Random;

public class HillClimbing {

    // Example objective function: f(x) = -(x-3)^2 + 9
    // The goal is to find the maximum value
    public static double objectiveFunction(double x) {
        return -Math.pow((x - 3), 2) + 9;
    }

    public static void main(String[] args) {
        Random rand = new Random();

        // Step 1: Initialize a random starting point
        double currentX = rand.nextDouble() * 10; // Random value between 0
and 10
        double stepSize = 0.1; // Step to move in each iteration
```

```
        double currentValue = objectiveFunction(currentX);

        System.out.println("Initial State: x = " + currentX + ", f(x) = " +
currentValue);

        while (true) {
            // Step 2: Generate neighboring states
            double nextX1 = currentX + stepSize;
            double nextX2 = currentX - stepSize;

            double nextValue1 = objectiveFunction(nextX1);
            double nextValue2 = objectiveFunction(nextX2);

            // Step 3: Find the better neighbor
            double bestNextX = currentX;
            double bestNextValue = currentValue;

            if (nextValue1 > bestNextValue) {
                bestNextX = nextX1;
                bestNextValue = nextValue1;
            }
            if (nextValue2 > bestNextValue) {
                bestNextX = nextX2;
                bestNextValue = nextValue2;
            }

            // Step 4: Check for improvement
            if (bestNextValue > currentValue) {
                currentX = bestNextX;
                currentValue = bestNextValue;
            } else {
                // No better neighbor found → peak reached
                break;
            }
        }

        System.out.println("Local Maximum found at: x = " + currentX);
        System.out.println("Maximum Value: f(x) = " + currentValue);
    }
}
```

Input/Output Examples:

Sample Output:

```
Initial State: x = 1.247, f(x) = 5.92
Local Maximum found at: x = 3.00
Maximum Value: f(x) = 9.00
```

Explanation:

- The algorithm starts with a random initial `x` (e.g., 1.247).
- It moves in the direction where the function value `f(x)` increases.
- When it can no longer find a better neighbor, it stops — having found the local maximum near `x = 3`.

Edge Case Example:
If the function has multiple peaks, the algorithm may stop at a **local maximum** rather than the **global maximum**, depending on the starting point.

Analysis of Code and Algorithm:

- **Logic:**
  The algorithm evaluates the current state and its neighboring states to iteratively move toward higher heuristic values until no better neighbor is found.
- **Heuristic Used:**
  The value of the objective function (f(x)) acts as the heuristic to be maximized.
- **Time Complexity:**
  O(n), where *n* is the number of iterations before convergence (depends on step size and function shape).
- **Space Complexity:**
  O(1), since only a few variables are stored.
- **Efficiency:**
  Works efficiently for simple continuous functions but can fail for complex, multi-modal functions due to local maxima traps.

Real-Life Applications:

1. **Route Optimization:**
   Used in navigation systems to find the shortest or most efficient route.
2. **Machine Learning Model Tuning:**
   Used for optimizing model parameters in feature selection and hyperparameter tuning.
3. **Robotics:**
   Applied in motion planning to reach optimal configurations efficiently.

Conclusion:

In this experiment, the **Hill Climbing Algorithm** was successfully implemented using Java. The algorithm demonstrated how iterative improvement can lead to a local optimum in a search space. Through this practical, the concept of heuristic-based optimization and the limitations of greedy local search methods were understood.