



|  |  |
| --- | --- |
| **Student Name** | **Arun Adhikari** |
| **SRN No** | 202100406 |
| **Roll No** | 31 |
| **Program** | **Department of Computer Engineering** |
| **Year** | 2024-2025 |
| **Division** | G |
| **Subject** | Artificial Intelligence |
| **Assignment No** | 3 |

**Assignment No:4**

**Title /Problem Statement**: Implement the Hill Climbing technique to solve the 8 puzzle problem**.**

**Theory:** The 8-puzzle problem is a well-known sliding puzzle consisting of a 3×3 grid with 8 numbered tiles and one blank space. The objective is to move the tiles into a goal state by sliding them in the empty space.

Hill climbing is a heuristic search algorithm used to solve optimization problems, including the 8-puzzle problem. It follows a greedy approach, always choosing the next move that appears to improve the current state.

Hill climbing is a local search algorithm that continuously moves toward a better state until no further improvement is possible.

**Key Characteristics**

* Starts from an initial state.
* Evaluates neighbouring states using a heuristic function.
* Moves to the best neighbouring state.
* Stops if no better neighbour exists (local optimum).

**2. Application to 8-Puzzle**

In the **8-puzzle**, hill climbing aims to minimize the number of misplaced tiles or the Manhattan distance (heuristic).

**Steps:**

1. **Initial State:** Start with a random configuration of tiles.
2. **Heuristic Function:** Evaluate each move based on a heuristic such as:
   * **Misplaced Tiles Heuristic:** Number of tiles not in their correct positions.
   * **Manhattan Distance Heuristic:** Sum of the distances of each tile from its goal position.
3. **Generate Successors:** Identify all possible moves (up, down, left, right).
4. **Choose the Best Move:** Select the move that results in the lowest heuristic value.
5. **Repeat Until Goal is Reached** or **No Better Moves Exist**.

**3. Problems with Hill Climbing in 8-Puzzle**

(a) Local Maxima:

* The algorithm may reach a state where no adjacent move improves the heuristic (gets stuck).

(b) Plateaus:

* Multiple neighbouring states have the same heuristic value, making it hard to choose the best move.

(c) Ridges:

* The best move requires a temporary increase in heuristic before improvement.

Solutions:

* Random Restart Hill Climbing: Restart from a new random state if stuck.
* Simulated Annealing: Introduce a probability of choosing worse moves early on to escape local maxima.
* *Best-First Search (A Algorithm):*\* Overcomes limitations by considering the cost of reaching the state along with the heuristic.

**Code:**

import random

import copy

def print\_puzzle(state):

    for row in state:

        print(" ".join(str(cell) if cell != 0 else " " for cell in row))

    print()

def get\_misplaced\_tiles(state, goal):

    misplaced = 0

    for i in range(3):

        for j in range(3):

            if state[i][j] != 0 and state[i][j] != goal[i][j]:

                misplaced += 1

    return misplaced

def get\_possible\_moves(state):

    moves = []

    x, y = [(i, row.index(0)) for i, row in enumerate(state) if 0 in row][0]

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]  # Right, Down, Left, Up

    for dx, dy in directions:

        new\_x, new\_y = x + dx, y + dy

        if 0 <= new\_x < 3 and 0 <= new\_y < 3:

            new\_state = copy.deepcopy(state)

            new\_state[x][y], new\_state[new\_x][new\_y] = new\_state[new\_x][new\_y], new\_state[x][y]

            moves.append(new\_state)

    return moves

def hill\_climbing(start, goal, max\_restarts=10):

    for \_ in range(max\_restarts):

        current\_state = start

        current\_cost = get\_misplaced\_tiles(current\_state, goal)

        print("Initial Heuristic Value:", current\_cost)

        print("\nPossible Moves and Heuristic Values:")

        while True:

            neighbors = get\_possible\_moves(current\_state)

            best\_move = None

            best\_cost = current\_cost

            for neighbor in neighbors:

                cost = get\_misplaced\_tiles(neighbor, goal)

                print\_puzzle(neighbor)

                print("Heuristic Value:", cost, "\n")

                if cost < best\_cost:

                    best\_move = neighbor

                    best\_cost = cost

            if best\_move is None or best\_cost >= current\_cost:

                break  # Stop if no better move

            current\_state = best\_move

            current\_cost = best\_cost

            print("Selected Best Heuristic Value:", current\_cost)

        if current\_state == goal:

            return current\_state, True  # Solution found

    return current\_state, False  # Stuck after max\_restarts

def get\_user\_input(prompt):

    print(prompt)

    values = list(map(int, input().split()))

    if len(values) != 9 or sorted(values) != list(range(9)):

        print("Invalid input! Please enter numbers 0-8 without duplicates.")

        return get\_user\_input(prompt)

    return [values[:3], values[3:6], values[6:9]]

def main():

    start\_state = get\_user\_input("Enter the initial state (space-separated 9 numbers from 0-8):")

    goal\_state = get\_user\_input("Enter the goal state (space-separated 9 numbers from 0-8):")

    print("\nInitial State:")

    print\_puzzle(start\_state)

    result, found = hill\_climbing(start\_state, goal\_state)

    print("Final State:")

    print\_puzzle(result)

    if found:

        print("Solution Found!")

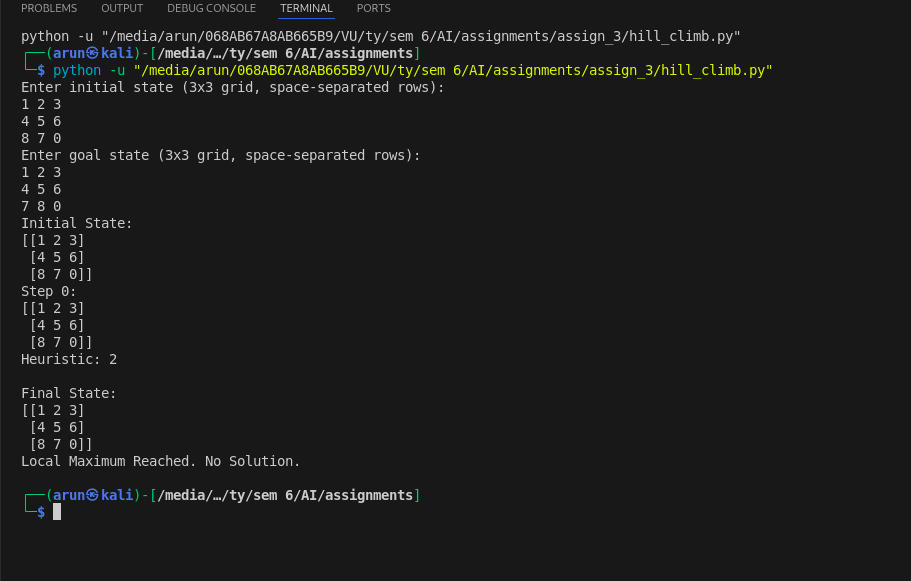
    else:

        print("Stuck at a local maximum even after multiple restarts. Try again!")

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

    main()

**Output:**



**Conclusion**: Hill climbing is an effective heuristic-based approach for solving the 8-puzzle, but it suffers from local optima problems. Enhancing it with random restarts or simulated annealing improves its performance**.**