Date:30.07.25

TASK:2

Implementation of Hill climbing algorithm for Heuristic search approach

Implementation of Hill climbing algorithm for Heuristic search approach using following constraints in python.

i. Create a function generating all neighbours of a solution
ii. Create a function calculating the length of a route
iii. Create a random solution generator
iv. Create a Travelling salesman problem

Tools- Python, Online Simulator - https://graphonline.ru/en/

PROBLEM STATEMENT: **CO1** **S3**

A hiker who is trying to reach the peak of a tall mountain. The hiker has a map of the mountain, but it only provides a partial view of the terrain. The goal of the hiker is to find the best path to the peak using the hill-climbing algorithm. The mountain is represented as a grid of cells, where each cell has a specific elevation value (height). The hiker starts at a random cell and can move to neighboring cells (up, down, left, or right). The elevation of the cell the hiker is currently at is called the current elevation.

The hiker needs to find the path that leads to the highest peak on the mountain. However, the hill-climbing algorithm has some limitations. Sometimes, it may get stuck in a local maximum, where the hiker reaches a peak that is not the highest overall

HILL CLIMBING

AIM

To implement the Hill Climbing algorithm using heuristic search in Python to help a hiker find a path to the highest reachable peak on a mountain.

ALGORITHM

1. Choose a starting cell randomly on the grid.
2. Check the elevation (height) of the current cell.
3. Look at the 4 neighbors (up, down, left, right).
4. Compare elevations:
    - Find the neighbor with the highest elevation.
    - Only choose it if it is higher than the current cell.
5. Move to that higher neighbor.
6. Repeat steps 2–5 until:
    - No neighbor is higher than the current cell.
7. Stop. You've reached a peak (maybe local, maybe global).

PROGRAM

Hill Climbing Algorithm (Heuristic Search)

```
import random
def get_neighbors(x, y, rows, cols):
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # up, down, left, right
    neighbors = []
    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if 0 <= nx < rows and 0 <= ny < cols:
            neighbors.append((nx, ny))
    return neighbors

def hill_climbing(grid, start):
    rows, cols = len(grid), len(grid[0])
    x, y = start
```

```
    current_path = [(x, y)]
    current_elevation = grid[x][y]

    while True:
        neighbors = get_neighbors(x, y, rows, cols)
        best_neighbor = None
        for nx, ny in neighbors:
            if grid[nx][ny] > current_elevation:
                                                if best_neighbor is None or grid[nx][ny] >
grid[best_neighbor[0]][best_neighbor[1]]:
                    best_neighbor = (nx, ny)

        if best_neighbor:
            x, y = best_neighbor
            current_path.append((x, y))
            current_elevation = grid[x][y]
        else:
            break  # No higher neighbor: local maximum reached

    return current_path, current_elevation

# Example mountain grid with elevation values
mountain = [
    [1, 2, 3, 4],
    [2, 3, 8, 5],
    [3, 4, 9, 6],
    [2, 5, 6, 7]
]

# Start at a random position
start_x = random.randint(0, len(mountain) - 1)
start_y = random.randint(0, len(mountain[0]) - 1)
start_position = (start_x, start_y)

print(f"Starting at position: {start_position} with elevation {mountain[start_x][start_y]}")

path, peak_elevation = hill_climbing(mountain, start_position)

print("Path to peak:")
print(path)
print(f"Peak elevation reached: {peak_elevation}")
```

OUTPUT

```
>>>
============ RESTART: C:/Users/student/Documents/27378 py/task 2.py ============
starting at position:(2, 1) with elevation 4
path to peak:
[(2, 1), (2, 2)]
peak elevation reached:9
>>>
```

RESULT

Thus, the Implementation of Hill climbing algorithm for Heuristic search approach to help a hiker find a path to the highest reachable peak on a mountain using python was successfully executed and output was verified.