

Name : Sufiyan Khan

D.O.P. : 02-08 -2023

RollNo:A12

Batch : A1

Class : T.E. AI-DS

## EXPERIMENT NO. 5

**Aim** : To implement Hill Climbing Algorithm to study the concept of search algorithms.

**Objective** : To understand and study one of the search algorithms used in artificial intelligence and optimization to find the best solution in a problem space.

**Theory** : Let us understand more about this algorithm step-by-step,

### 1. What is a searching algorithm in AI?

A search algorithm in the context of artificial intelligence (AI) refers to a systematic and methodical approach used to find a solution to a problem within a search space. The goal of a search algorithm is to navigate through a set of possible states or configurations to locate a state that satisfies specific criteria or criteria. Search algorithms are fundamental in various AI applications, including problem-solving, game playing, route planning, and optimization.

### 2. Explain Hill Climbing Algorithm.

Hill climbing is a heuristic search algorithm used in artificial intelligence and optimization to find the best solution in a problem space. The basic idea behind the hill climbing algorithm is to start with an arbitrary solution and iteratively make small incremental changes to the current solution, moving in the direction that improves the solution's quality.

### 3. How does the Hill Climbing Algorithm work?

Here's how the hill climbing algorithm typically works:

→ Initialization: Start with an initial solution or state.

→ Evaluation: Evaluate the quality or fitness of the current solution. The quality is usually determined by a predefined objective function or evaluation criteria.

→ Neighbor Generation: Generate neighboring solutions by making small modifications to the current solution. These modifications could involve changing one element or variable at a time.

→Selection: Select the neighboring solution that improves the quality the most. If no neighbor is better, the algorithm terminates.

→Update: Replace the current solution with the selected neighboring solution and return to step 2. This process is repeated iteratively until no further improvements can be made.

### Example :

#### Program :

```
import numpy as np
```

```
def find_neighbours(state, landscape):
```

```
    neighbours = []
```

```
    dim = landscape.shape
```

```
    if state[0] != 0:
```

```
        neighbours.append((state[0] - 1, state[1]))
```

```
    if state[0] != dim[0] - 1:
```

```
        neighbours.append((state[0] + 1, state[1]))
```

```
    if state[1] != 0:
```

```
        neighbours.append((state[0], state[1] - 1))
```

```
    if state[1] != dim[1] - 1:
```

```
        neighbours.append((state[0], state[1] + 1))
```

```
    if state[0] != 0 and state[1] != 0:
```

```
        neighbours.append((state[0] - 1, state[1] - 1))
```

```
    if state[0] != 0 and state[1] != dim[1] - 1:
```

```
    neighbours.append((state[0] - 1, state[1] + 1))
```

```
if state[0] != dim[0] - 1 and state[1] != 0:
```

```
    neighbours.append((state[0] + 1, state[1] - 1))
```

```
if state[0] != dim[0] - 1 and state[1] != dim[1] - 1:
```

```
    neighbours.append((state[0] + 1, state[1] + 1))
```

```
return neighbours
```

```
def hill_climb(curr_state, landscape):
```

```
    neighbours = find_neighbours(curr_state, landscape)
```

```
    bool
```

```
    ascended = False
```

```
    next_state = curr_state
```

```
    for neighbour in neighbours:
```

```
        if landscape[neighbour[0]][neighbour[1]] > landscape[next_state[0]][next_state[1]]:
```

```
            next_state = neighbour
```

```
            ascended = True
```

```
    return ascended, next_state
```

```
def _main_():
```

```
    landscape = np.random.randint(1, high=60, size=(8, 8))
```

```
    print(landscape)
```

```
    start_state = (2, 4)
```

```
    current_state = start_state
```

```

count = 1

ascending = True

while ascending:

    print("\nStep ", count, ":")

    print("Current state coordinates: ", current_state)

    print("Current state value: ", landscape[current_state[0]][current_state[1]])

    count += 1

    ascending, current_state = hill_climb(current_state, landscape)

print("\nStep ", count, ":")

print("Optimization objective reached.")

print("Final state coordinates: ", current_state)

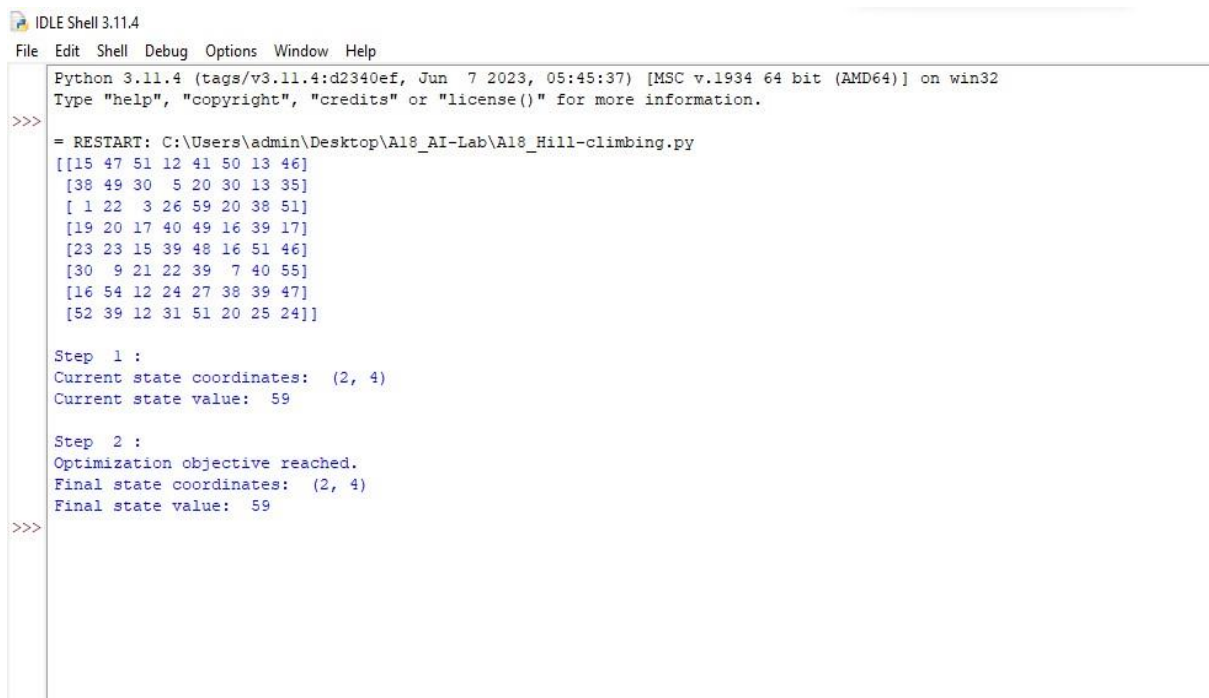
print("Final state value: ", landscape[current_state[0]][current_state[1]])

```

`_main_()`

### **Output :**

(i) This comes out to be the graph that has been generated for 8 x 8 matrix so the output comes out to be,



```

IDLE Shell 3.11.4
File Edit Shell Debug Options Window Help
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\admin\Desktop\AI8_AI-Lab\AI8_Hill-climbing.py
[[15 47 51 12 41 50 13 46]
 [38 49 30 5 20 30 13 35]
 [ 1 22 3 26 59 20 38 51]
 [19 20 17 40 49 16 39 17]
 [23 23 15 39 48 16 51 46]
 [30 9 21 22 39 7 40 55]
 [16 54 12 24 27 38 39 47]
 [52 39 12 31 51 20 25 24]]

Step 1 :
Current state coordinates: (2, 4)
Current state value: 59

Step 2 :
Optimization objective reached.
Final state coordinates: (2, 4)
Final state value: 59
>>>

```

(ii) We have taken the coordinates as (2,4) which in the generated matrix is 59, so we check for any maximum value if any, to proceed and to print the final state value.

(iii) The final state value thus comes out to be 59 after checking the neighbours – top, bottom, left, right and the diagonal elements - for any maximum value and there is none so the final value is given to us as 59.

15	47	51	12	41	50	13	46
38	49	30	5	20	30	13	35
1	22	3	26	59	20	38	51
19	20	17	40	49	16	39	17
23	23	15	39	48	16	51	46
30	9	21	22	39	7	40	55
16	54	12	24	27	38	39	47
52	39	12	31	51	20	25	24

Now, if we take a different coordinate in the same program we will get a different optimal value,

(i) So, here we take (1,4) which in the matrix is 12, so we check for any maximum value if any, to proceed and to print the final state value.

(ii) We get a value greater than 12 at the coordinate (2,3) and that value is 55, we get this value by checking the neighbours of 12.

(iii) We move to the node 55 and check if there's any value greater than 55 in its neighbourhood, we find none. The output comes out to be,

```

===== RESTART: C:\Users\admin\Desktop\AI8_AI-Lab\AI8_Hill-climbing.py ===
[[42 39 36  6 31 45 25  5]
 [47 52  1 18 12 41 52  8]
 [49 15 27 55 38 52 58 33]
 [32 59 28 40 39 57 26 19]
 [22 52 32  9 43 25  7 52]
 [19 28 36 48 44 44 26 26]
 [51 57  5 41  1 42 23 48]
 [16 40 49 27 47 39 57 42]]

Step 1 :
Current state coordinates: (1, 4)
Current state value: 12

Step 2 :
Current state coordinates: (2, 3)
Current state value: 55

Step 3 :
Optimization objective reached.
Final state coordinates: (2, 3)
Final state value: 55

```

(iv) The final state value thus comes out to be 55 after checking the neighbours – top , bottom , left , right and the diagonal elements - for any maximum value and there is none so the final value is given to us as 55.

42	39	36	6	31	45	25	5
47	52	1	18	12	41	52	8
49	15	27	55	38	52	58	33
32	59	28	40	39	57	26	19
22	52	32	9	43	25	7	52
19	28	36	48	44	44	26	26
51	57	5	41	1	42	23	48
16	40	49	27	47	39	57	42

Hence we can see how the Hill Climbing Algorithm works with the examples that are solved above.

**CONCLUSION :**

Hence in this experiment, we studied the concept of search algorithms in AI through Hill Climbing Algorithm and successfully implemented it through various examples with a matrix. This thus satisfies the aim of the experiment.