A.Y. 2022-2023

Subject: Artificial Intelligence **SAP ID: 60004220253 – Devansh Mehta**

Experiment No. 04

Aim: Implementation of Hill Climbing Algorithm

Theory:

Hill climbing is a local search algorithm used for mathematical optimization problems. It starts with an arbitrary solution to a problem and iteratively makes small improvements to it, moving towards a better solution. At each iteration, thealgorithm evaluates the neighboring solutions and selects the one that improves the current solution the most. This process continues until no further improvements can be made, and the algorithm reaches a local optimum.

Here is a brief overview of the hill climbing algorithm:

1. **Initialization**: Start with an initial solution to the problem.
2. **Evaluation**: Evaluate the current solution by calculating its fitness or cost.
3. **Neighbor Generation**: Generate neighboring solutions by making small modifications to the current solution. Neighbors are potential solutions thatare close to the current one.
4. **Selection**: Choose the best neighboring solution based on its fitness or cost.
5. **Termination**: Check if the stopping criteria are met. This could be a maximum number of iterations, reaching a satisfactory solution, or other criteria.
6. **Update**: If a better solution is found, update the current solution with the selected neighbor and repeat from step 2.

Hill climbing has several variants, including steepest ascent hill climbing (wherethe steepest ascent neighbor is always selected) and steepest descent hill climbing(where the steepest descent neighbor is always selected).

Code:

```
import copy

visited_states = []

def gn(curr_state,prev_heu,goal_state):
    global visited_states
    state = copy.deepcopy(curr_state)
    for i in range(len(state)):
        temp = copy.deepcopy(state)
        if len(temp[i]) > 0:
            elem = temp[i].pop()
            for j in range(len(temp)):
                temp1 = copy.deepcopy(temp)
```

```
            if j != i:
                temp1[j] = temp1[j] + [elem]
                if (temp1 not in visited_states):
                    curr_heu=heuristic(temp1,goal_state)
                    if curr_heu>prev_heu:
                        child = copy.deepcopy(temp1)
                        return child
    return 0

def heuristic(curr_state,goal_state):
    goal_=goal_state[3]
    val=0
    for i in range(len(curr_state)):
        check_val=curr_state[i]
        if len(check_val)>0:
            for j in range(len(check_val)):
                if check_val[j]!=goal_[j]:
                    # val-=1
                    val-=j
                else:

                    # val+=1
                    val+=j
    return val
def sln(init_state,goal_state):
    global visited_states
    if (init_state == goal_state):
        print (goal_state)
        print("solution found!")
        return

    current_state = copy.deepcopy(init_state)
    while(True):
```

```
        visited_states.append(copy.deepcopy(current_state))
        print(current_state)
        prev_heu=heuristic(current_state,goal_state)
        child = gn(current_state,prev_heu,goal_state)
        if child==0:
            print("Final state - ",current_state)
            return

        current_state = copy.deepcopy(child)

 def main():
    global visited_states
    initial = [[],[],[],['B','C','D','A']]
    goal = [[],[],[],['A','B','C','D']]
    sln(initial,goal)
main()
```

```
[[], [], [], ['B', 'C', 'D', 'A']]
[['A'], [], [], ['B', 'C', 'D']]
[['A', 'D'], [], [], ['B', 'C']]
[['A'], ['D'], [], ['B', 'C']]
[['A'], ['D'], ['C'], ['B']]
[['A', 'B'], ['D'], ['C'], []]
[['A', 'B', 'C'], ['D'], [], []]
[['A', 'B', 'C', 'D'], [], [], []]
Final state -  [['A', 'B', 'C', 'D'], [], [], []]
```

Conclusion: In conclusion, Hill climbing is a local search algorithm that iteratively improves a solution by selecting the best neighboring solution. Conclusions depend on factors such as sensitivity to the initial solution, the presence of local optima, and the efficiency of local exploration. The algorithm may find satisfactory solutions quickly but is susceptible to getting stuck in localoptima.