Shreyas Rami
60004190105
TE B
AI

Experiment 4

Aim:

Program to implement Local Search algorithm: Hill climbing search.

Theory:

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbour has a higher value. This algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance travelled by the salesman. It is also called greedy local search as it only looks to its good immediate neighbour state and not beyond that. In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state. A node of hill climbing algorithm has two components which are state and value. It is mostly used when a good heuristic is available.

Following are some main features of Hill Climbing Algorithm:

Generate and Test variant: Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.

Greedy approach: Hill-climbing algorithm search moves in the direction which optimizes the cost.

No backtracking: It does not backtrack the search space, as it does not remember the previous states.

Types of Hill Climbing Algorithm are:

1. Simple hill Climbing
2. Steepest-Ascent hill-climbing
3. Stochastic hill Climbing

Algorithm

Step 1: Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.

Step 2: Loop until the solution state is found or there are no new operators present which can be applied to the current state.

a) Select a state that has not been yet applied to the current state and apply it to produce a new state.

b) Perform these to evaluate new state:

1. If the current state is a goal state, then stop and return success.

2. If it is better than the current state, then make it current state and proceed further.

3. If it is not better than the current state, then continue in the loop until a solution is found.

Step 3: Exit.

Code:

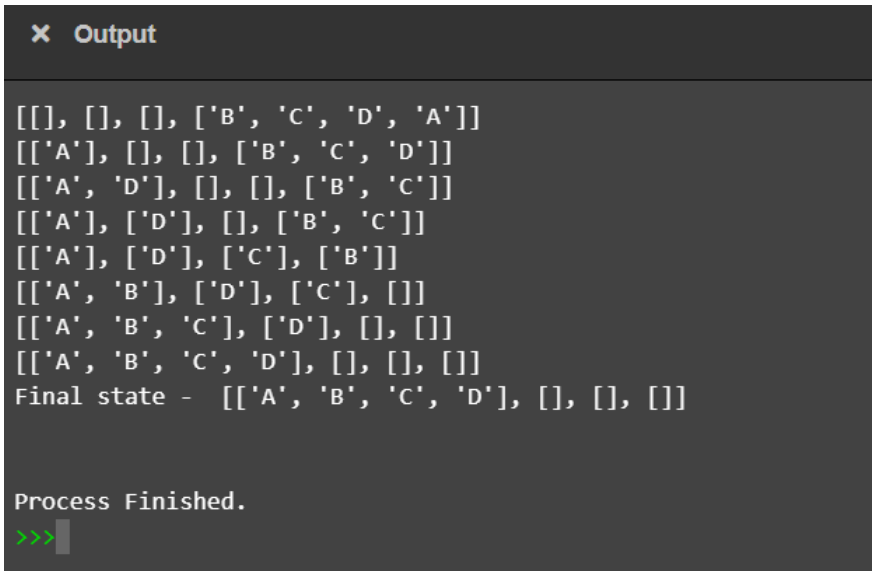
```
import copy

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

hc()
```

Output:



Conclusion:

Hill Climbing algorithm is good in solving the optimization problem while using only limited computation power.