

## **LAB-D: WRITE A PROGRAM TO SOLVE THE BLOCK WORLD PROBLEM USING HILL CLIMBING SEARCH.**

**OBJECTIVE:** To implement block world problem using hill search.

### **THEORY:**

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value. It stops when it reaches a “peak” where no neighbor has higher value. This algorithm is considered to be one of the simplest procedures for implementing heuristic search. The hill climbing comes from that idea if you are trying to find the top of the hill and you go up direction from wherever you are. This heuristic combines the advantages of both depth first and breadth first searches into a single method. simulating

The name hill climbing is derived from the situation of a person climbing the hill. The person will try to move forward in the direction of at the top of the hill. His movement stops when it reaches the peak of hill and no peak has higher value of heuristic function than this. Hill climbing uses knowledge about the local terrain, providing a very useful and effective heuristic for eliminating much of the unproductive search space. It is a branch by a local evaluation function. The hill climbing is a variant of generate and test in which direction the search should proceed. At each point in the search path, a successor node that appears to reach for exploration

### **ALGORITHM FOR HILL SEARCH:**

**Step 1:** Evaluate the starting state. If it is a goal state then stop and return success.

**Step 2:** Else, continue with the starting state as considering it as a current state.

**Step 3:** Continue step-4 until a solution is found i.e. until there are no new states left to be in the current state.

#### **Step 4:**

- a) Select a state that has not been yet applied to the current state and apply it to produce a new state.
- b) Procedure to evaluate a new state.
  - i. If the current state is a goal state, then stop and return success.
  - ii. If it is better than the current state, then make it current state and proceed further.
  - iii. If it is not better than the current state, then continue in the loop until a solution is found.

**Step 5:** Exit.

### **Heuristic Function -**

**H(n)** : Number of misplaced blocks in the current state as compared to the goal state.

### **Steps -**

1. Evaluate initial state. If its equal to goal state - (found) return.
2. Loop until a solution is found or next states cause no change in the heuristic value.
  - a). Try going to next states one by one, if any better state found, move to next better state without considering other possibilities.
  - (b). If no better next state found, stop and return

## PROGRAM:

"""

Tilak Parajuli

### Question -

Blocks World problem using Simple Hill Climbing.

### Heuristic Function -

$H(n)$  : Number of misplaced blocks in the current state as compared to the goal state.

### Steps -

1. Evaluate initial state. If its equal to goal state - (found) return.
2. Loop until a solution is found or next states cause no change in the heuristic value.
  - (a). Try going to next states one by one, if any better state found, move to next better state without considering other possibilities.
  - (b). If no better next state found, stop and return

"""

import copy

visited\_states = []

# heuristic fn - number of misplaced blocks as compared to goal state

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

# generate next possible solution for the current state

def generate\_next(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 a better state than previous state of
                found
                if curr_heu>prev_heu:
                    child = copy.deepcopy(temp1)
                    return child

# no better soln than current state is possible
return 0

```

```

def solution_(init_state,goal_state):
global visited_states

```

```

# checking if initial state is already the final state
if (init_state == goal_state):
    print (goal_state)
    print("solution found!")
    Return

```

```

current_state = copy.deepcopy(init_state)

```

```

# loop while goal is found or no better optimal solution is possible
while(True):

```

```

    # add current state to visited to avoid repetition
    visited_states.append(copy.deepcopy(current_state))

```

```

    print(current_state)
    prev_heu=heuristic(current_state,goal_state)

```

```

    # generate possible better child from current state
    child = generate_next(current_state,prev_heu,goal_state)

```

```

    # No more better states are possible
    if child==0:
        print("Final state - ",current_state)
        return

```

```

    # change current state to child

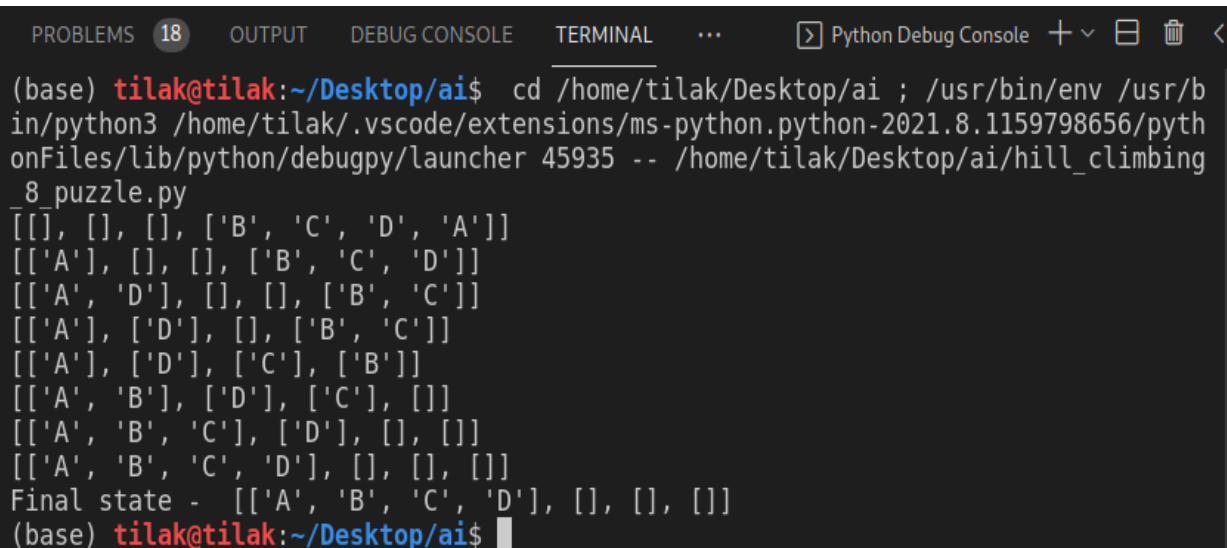
```

```
current_state = copy.deepcopy(child)
```

```
def solver():  
    # maintaining a global visited to save all visited and avoid repetition & infinite  
    loop condition  
    global visited_states  
    # inputs  
    init_state = [[],[],[],['B','C','D','A']]  
    goal_state = [[],[],[],['A','B','C','D']]  
    # goal_state = [[],[],[],['A','D','C','B']]  
    solution_(init_state,goal_state)
```

```
solver()
```

## OUTPUT:



```
PROBLEMS 18 OUTPUT DEBUG CONSOLE TERMINAL ... Python Debug Console + v [icon] [icon] [icon]  
(base) tilak@tilak:~/Desktop/ai$ cd /home/tilak/Desktop/ai ; /usr/bin/env /usr/b  
in/python3 /home/tilak/.vscode/extensions/ms-python.python-2021.8.1159798656/pyth  
onFiles/lib/python/debugpy/launcher 45935 -- /home/tilak/Desktop/ai/hill_climbing  
_8_puzzle.py  
[[], [], [], ['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'], [], [], []]  
(base) tilak@tilak:~/Desktop/ai$
```

## CONCLUSION:

Hill climbing is a very resourceful technique used in solving huge computational problems. It can help establish the best solution for problems. This technique has the potential of revolutionizing optimization within artificial intelligence.

In the future, technological advancement to the hill climbing technique will solve diverse and unique optimization problems with better advanced features.