AI Lab 2 – Report

180010019 - Karan Sharma

180010008-Balsher Singh

Input and Output

Command to run- 'python 3.py input.txt'

- Input.txt example

```
1              # '1' for best first search, '2' for hill climbing
b c a          #stack 1 of start state
e d            #stack 2 of start state
f              #stack 3 of start state
a b d          #stack 1 of goal state
f e            #stack 2 of goal state
c              #stack 3 of goal state
```

- Output.txt example

  Length of path

  Number of states explored

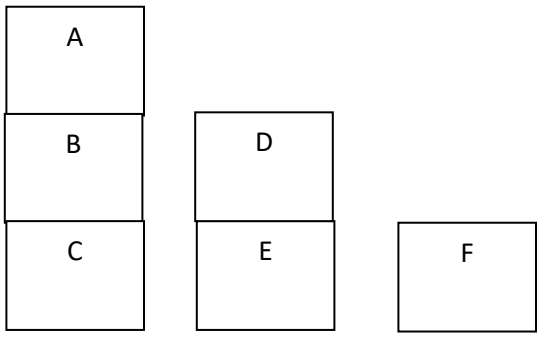
  path of state from start state to goal state (In case of hill climbing stuck in local maxima it will print "stuck in local maxima"

1. **Description about domain (State space, start node, goal node)**

   - Three different stacks of block where top block of any stack can be moved to top of any other remaining stacks to achieve a new state.
   - Start state and goal state are given as input
   - We have to arrange the block in such a way to achieve the goal state
   - In code, State is a Class, their attribute stack1, stack2, stack3 represented 3 stacks of block. Parent attribute is for determining path and heuristic attribute is for storing heuristic value.
   - Example of a state:

     In code following state is represented as

     Stack1 = [‘C’,’B’,’A’], Stack2 = [‘E’,’D’], Stack3 = [‘F’]



2. MoveGen

```
MoveGen(state):
    Initialize empty list for neighbour
    If stack1:
        New state = copy of state
        Pop from stack1 of new state and append in stack2
        Neighbout.append(new state)
        New state = copy of state
        Pop from stack1 of new state and append in stack3
        Neighbout.append(new state)
    If stack2:
        New state = copy of state
        Pop from stack2 of new state and append in stack1
        Neighbout.append(new state)
        New state = copy of state
        Pop from stack2 of new state and append in stack3
        Neighbout.append(new state)
    If stack3:
        New state = copy of state
        Pop from stack3 of new state and append in stack2
        Neighbout.append(new state)
        New state = copy of state
        Pop from stack3 of new state and append in stack1
        Neighbout.append(new state)
    Return neighbour
```

3. Goal Test

Goal Test (state, goal state):

If stack1 of state != stack1 of goal state:

Return False

If stack1 of state != stack1 of goal state:

Return False

If stack1 of state != stack1 of goal state:

Return False

Return True

4. Heuristic

Priority key of state ‘s’ in Min priority queue = heuristic of goal state – heuristic of state ‘s’

*** We are doing it because min priority queue is used, so we want to select that node which has minimum priority key***

- **Heuristic 1**

  +1 for block if it is in correct stack **and** on the top of correct block

  -1 for block otherwise

  Example:

  Start state – [a, b, f], [d], [c, e] Goal state – [b, f], [c, a], [d, e]

  Heuristic of start state = -1-1+1-1-1-1 = -4

  Heuristic of goal state = +6

- **Heuristic 2**

  +1 x (level of block) if block is in correct stack and on the top of correct structure i.e., all the block below it must be in correct form

  -1 x (level of block) otherwise

  * ground block has level 1

Example:

Start state – [a, b, f], [d], [c, e] Goal state – [b, f], [c, a], [d, e]

Heuristic of start state = -1-2-3-1-1-2 = -10

Heuristic of goal state = +9

- **Heuristic 3**

  +1 x (height of stack - level of block) if block is in correct stack and on the top of correct structure i.e., all the block below it must be in correct form

  -1 x (height of stack - level of block) otherwise

  ** level of ground block is 1

Start state – [a, b, f], [d], [c, e] Goal state – [b, f], [c, a], [d, e]

Heuristic of start state = -3-2-1-1-2-1 = -10

Heuristic of goal state = +9

5. Best first search

Input	Heuristic 1	Heuristic 2	Heuristic 3
Start state [a] [] [] Goal state [] [] [a]	No. of states in path: 2 No. of explored states: 2	No. of states in path: 2 No. of explored states: 2	No. of states in path: 2 No. of explored states: 2
Start state [a, b, c] [d] [] Goal state [d, c] [a] [b]	No. of states in path: 8 No. of explored states: 16	No. of states in path: 10 No. of explored states: 16	No. of states in path: 10 No. of explored states: 14

Start state [e, a, c] [b, d] [f] Goal state [f, d] [a] [c, b, e]	No. of states in path: 15 No. of explored states: 28	No. of states in path: 20 No. of explored states: 101	No. of states in path: 22 No. of explored states: 86
Start state [e, b, f] [d, a] [c] Goal state [a, d, b] [e, f, c] []	No. of states in path: 23 No. of explored states: 59	No. of states in path: 47 No. of explored states: 323	No. of states in path: 53 No. of explored states: 365
Start state [3, 4, 5] [2, 6, 1] [] Goal state [4, 2] [6, 1, 5, 3] []	No. of states in path: 14 No. of explored states: 34	No. of states in path: 51 No. of explored states: 892	No. of states in path: 57 No. of explored states: 1185

Observation

- Heuristic 1 is giving best and fastest solution. It penalizes irrespective of height of block with -1 which proves to be efficient where number of stacks of block are limited i.e., 3.
- Heuristic 2 and 3 are performing around equally with smaller input but with large input heuristic 2 is doing better. It indicates that if a block placed correctly on higher level should contribute more than block which are on lower level.

6. Comparing Hill Climb and Best first search

Input	Best first search	Hill climb
Start state [3, 4, 5] [2, 6, 1] [] Goal state [4, 2] [6, 1, 5, 3] []	**Heuristic 1:** No. of explored states: 34 Time: 16 ms Path: Found **Heuristic 2:** No. of explored states: 892 Time: 2.5 s Path: Found **Heuristic 3:** No. of explored states: 1185 Time: 3.5 s Path: Found	**Heuristic 1:** No. of explored states: 3 Time: 1 ms Path: Not Found **Heuristic 2:** No. of explored states: 4 Time: 1 ms Path: Not Found **Heuristic 3:** No. of explored states: 4 Time: 1 ms Path: Not Found

Conclusion: Best first search is able to find solution but hill climb is stuck in local maxima.