

18/10/24

LAB-2

Q) Implement a Vacuum cleaner agent using python

ans)

function

Vacuum()

goal-state = { 'A': '0', 'B': '0' }

cost = 0

Input location-input, status-input, status-input-complement

Print "Initial condition:", goal-state

If location-input == 'A' then

If status-input == '1' then

goal-state['A'] = '0'

cost += 1

If status-input-complement == '1' then

goal-state['B'] = '0'

cost += 2 // Move and clean B

Else

If status-input == '1' then

goal-state['B'] = '0'

cost += 1

If status-input-complement == '1' then

goal-state['A'] = '0'

cost += 2 // Move and clean A

Print "Goal state:", goal-state

Print "cost:", cost

End function

OUTPUT:

Locations : A-0 , B-1

Enter location of Vacuum : 1 (B)

Enter status of room (0-clean, 1-dirty) : 1

Enter status of other room : 0

Initial condition : {A:0, B:1}

Vacuum is placed in B

Location B is dirty

Cost for cleaning : 1

Location B becomes

Location A already clean

Final state : {A:0, B:0}

Cost : 1

18/10/24

Q) Implement 8 puzzle problem using BFS algorithm

ans) Algorithm:
 Let fringe be a list containing the initial state
 Loop
 If the fringe is empty return failure
 Node ← remove-first(fringe)
 If Node is a goal:
 then return path from initial state to Node
 also generate all successors of Node and
 add generated nodes to the back of fringe
 End loop

consider initial and final state

1 2 3		1 2 3
4 5 6	↔	4 5 6
0 7 8		7 8 0

initial

final

1	2	3
4	5	6
0	7	8

1	2	3
0	5	6
4	7	8

1	2	3
4	5	6
7	0	8

0	2	3
1	5	6
4	7	8

1	2	3
5	0	6
4	7	8

1	2	3
4	5	6
7	8	0

1	2	3
4	0	6
7	5	8

final state

Q) Implement 8 puzzle problem using DFS algorithm

ans) Algorithm:

Let fringe be a list containing the initial state

Loop

If fringe is empty return failure

Node ← remove-first (fringe)

If Node is a goal:

then return path from initial state to Node

else generate all successors of Node and

add generated nodes to front of fringe

End loop

Initial state -

1	2	3
4	5	6
0	7	8

1 2 3

0 5 6

4 7 8

0 2 3

1 5 6

4 7 8

2 0 3

1 5 6

4 7 8

1 2 3

4 5 6

7 8 0

} final state

```

def vacuum_world():
    # Initialize goal state: 0 indicates Clean and 1 indicates Dirty
    goal_state = {'A': '0', 'B': '0'}
    cost = 0

    # User input for vacuum location and status
    location_input = input("Enter location of vacuum (A/B): ").strip().upper()
    status_input = input(f"Enter status of {location_input} (0 for Clean, 1 for Dirty): ").strip()
    status_input_complement = input(f"Enter status of other room ({'B' if location_input == 'A' else 'A'}): ").strip()

    print("Initial Location Condition: " + str(goal_state))

    if location_input == 'A':
        print("Vacuum is placed in Location A")

        if status_input == '1': # Location A is Dirty
            print("Location A is Dirty.")
            # Clean A
            goal_state['A'] = '0'
            cost += 1 # Cost for cleaning
            print("Cost for CLEANING A: " + str(cost))
            print("Location A has been Cleaned.")

        if status_input_complement == '1': # If B is Dirty
            print("Location B is Dirty.")
            print("Moving right to Location B.")
            cost += 1 # Cost for moving right
            print("Cost for moving RIGHT: " + str(cost))
            # Clean B
            goal_state['B'] = '0'
            cost += 1 # Cost for cleaning
            print("Cost for CLEANING B: " + str(cost))
            print("Location B has been Cleaned.")
        else:
            print("Location B is already clean.")

```



```

34:         else:
35:             print("Location B is already clean.")
36:     else:
37:         print("Location A is already clean.")
38:
39:         if status_input_complement == '1': # If B is Dirty
40:             print("Location B is Dirty.")
41:             print("Moving right to Location B.")
42:             cost += 1 # Cost for moving right
43:             print("Cost for moving RIGHT: " + str(cost))
44:             # Clean B
45:             goal_state['B'] = '0'
46:             cost += 1 # Cost for cleaning
47:             print("Cost for CLEANING B: " + str(cost))
48:             print("Location B has been Cleaned.")
49:         else:
50:             print("Location B is already clean.")
51:
52:     else: # Vacuum is placed in Location B
53:         print("Vacuum is placed in Location B")
54:
55:         if status_input == '1': # Location B is Dirty
56:             print("Location B is Dirty.")
57:             # Clean B
58:             goal_state['B'] = '0'
59:             cost += 1 # Cost for cleaning
60:             print("Cost for CLEANING B: " + str(cost))
61:             print("Location B has been Cleaned.")
62:
63:         if status_input_complement == '1': # If A is Dirty
64:             print("Location A is Dirty.")
65:             print("Moving left to Location A.")
66:             cost += 1 # Cost for moving left
67:             print("Cost for moving LEFT: " + str(cost))

```

```

67         print("Cost for moving LEFT: " + str(cost))
68         # Clean A
69         goal_state['A'] = '0'
70         cost += 1 # Cost for cleaning
71         print("Cost for CLEANING A: " + str(cost))
72         print("Location A has been Cleaned.")
73     else:
74         print("Location A is already clean.")
75 else:
76     print("Location B is already clean.")
77
78     if status_input_complement == '1': # If A is Dirty
79         print("Location A is Dirty.")
80         print("Moving left to Location A.")
81         cost += 1 # Cost for moving left
82         print("Cost for moving LEFT: " + str(cost))
83         # Clean A
84         goal_state['A'] = '0'
85         cost += 1 # Cost for cleaning
86         print("Cost for CLEANING A: " + str(cost))
87         print("Location A has been Cleaned.")
88     else:
89         print("Location A is already clean.")
90
91     # Done cleaning
92     print("GOAL STATE: ")
93     print(goal_state)
94     print("Performance Measurement: " + str(cost))
95
96 print("Tanish M V")
97 print("1BM22CS302")
98 vacuum_world()

```

Tanish M V

1BM22CS302

Enter location of vacuum (A/B): A

Enter status of A (0 for Clean, 1 for Dirty): 1

Enter status of other room (B): 0

Initial Location Condition: {'A': '0', 'B': '0'}

Vacuum is placed in Location A

Location A is Dirty.

Cost for CLEANING A: 1

Location A has been Cleaned.

Location B is already clean.

GOAL STATE:

{'A': '0', 'B': '0'}

Performance Measurement: 1

1 #8 puzzle using DFS

2 class PuzzleState:

3 def __init__(self, state, parent=None):

4 self.state = state

5 self.parent = parent

6
7 def __str__(self):

8 return "\n".join([str(self.state[i:i+3]) for i in range(0, 9, 3)])

9
10 def get_possible_moves(self):

11 moves = []

12 zero_pos = self.state.index(0)

13
14 directions = [

15 (-3, "Up"),

16 (3, "Down"),

17 (-1, "Left"),

18 (1, "Right")

19]

20
21 for direction, move in directions:

22 new_pos = zero_pos + direction

23 if 0 <= new_pos < 9:

24 if (move == "Left" and zero_pos % 3 == 0) or (move == "Right" and new_pos % 3 == 0):
25 continue

26 new_state = self.state[:]

27 new_state[zero_pos], new_state[new_pos] = new_state[new_pos], new_state[zero_pos]

28 moves.append(new_state)

29
30 return moves

31
32 def is_goal_state(self):

33 return self.state == [1, 2, 3, 4, 5, 6, 7, 8, 0]

34
35
36 def dfs(initial_state, goal_state):

37 stack = [PuzzleState(initial_state)]


```

38     visited.add(tuple(initial_state))
39
40     while stack:
41         current_state = stack.pop()
42
43         if current_state.is_goal_state():
44             solution = []
45             while current_state:
46                 solution.append(current_state.state)
47                 current_state = current_state.parent
48             solution.reverse()
49             return solution
50
51         for next_state in current_state.get_possible_moves():
52             if tuple(next_state) not in visited:
53                 visited.add(tuple(next_state))
54                 stack.append(PuzzleState(next_state, current_state))
55
56     return None
57
58
59 def print_solution(solution):
60     if solution:
61         print("Solution:")
62         for state in solution:
63             print("\n".join([str(state[i:i+3]) for i in range(0, 9, 3)]))
64             print()
65     else:
66         print("No solution found.")
67
68
69 initial_state = [1, 2, 3, 4, 0, 5, 7, 8, 6]
70 goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
71

```



```
68
69
70 initial_state = [1, 2, 3, 4, 0, 5, 7, 8, 6]
71 goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
72
73 print("Tanish M V")
74 print("1BM22CS302")
75 print("8 puzzle using DFS:")
76
77 solution = dfs(initial_state, goal_state)
78
79 print_solution(solution)
```


Tanish M V

1BM22CS302

8 puzzle using DFS:

Solution:

[1, 2, 3]

[4, 0, 5]

[7, 8, 6]

[1, 2, 3]

[4, 5, 0]

[7, 8, 6]

[1, 2, 3]

[4, 5, 6]

[7, 8, 0]

```
1 #8 puzzle using bfs
2 from collections import deque
3
4 class PuzzleState:
5     def __init__(self, state, parent=None):
6         self.state = state
7         self.parent = parent
8
9     def __str__(self):
10         return "\n".join([str(self.state[i:i+3]) for i in range(0, 9, 3)])
11
12     def get_possible_moves(self):
13         moves = []
14         zero_pos = self.state.index(0)
15
16         directions = [
17             (-3, "Up"),
18             (3, "Down"),
19             (-1, "Left"),
20             (1, "Right")
21         ]
22
23         for direction, move in directions:
24             new_pos = zero_pos + direction
25             if 0 <= new_pos < 9:
26                 if (move == "Left" and zero_pos % 3 == 0) or (move == "Right" and new_pos % 3 == 0):
27                     continue
28                 new_state = self.state[:]
29                 new_state[zero_pos], new_state[new_pos] = new_state[new_pos], new_state[zero_pos]
30                 moves.append(new_state)
31
32         return moves
33
```



```

31         return moves
32
33     def is_goal_state(self):
34         return self.state == [1, 2, 3, 4, 5, 6, 7, 8, 0]
35
36
37 def bfs(initial_state, goal_state):
38     queue = deque([PuzzleState(initial_state)])
39     visited = set()
40     visited.add(tuple(initial_state))
41
42     while queue:
43         current_state = queue.popleft()
44
45         if current_state.is_goal_state():
46             solution = []
47             while current_state:
48                 solution.append(current_state.state)
49                 current_state = current_state.parent
50             solution.reverse()
51             return solution
52
53         for next_state in current_state.get_possible_moves():
54             if tuple(next_state) not in visited:
55                 visited.add(tuple(next_state))
56                 queue.append(PuzzleState(next_state, current_state))
57
58     return None
59
60
61 def print_solution(solution):
62     if solution:

```

```
62 def print_solution(solution):
63     if solution:
64         print("Solution Path (from initial to goal):")
65         for state in solution:
66             print("\n".join([str(state[i:i+3]) for i in range(0, 9, 3)]))
67             print()
68     else:
69         print("No solution found.")
70
71
72 initial_state = [1, 2, 3, 4, 0, 5, 7, 8, 6]
73 goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
74
75 print("Tanish M V")
76 print("1BM22CS302")
77 print("8 puzzle using BFS:")
78
79 solution = bfs(initial_state, goal_state)
80
81 print_solution(solution)
82
```


Tanish M V

1BM22CS302

8 puzzle using BFS:

Solution Path (from initial to goal):

[1, 2, 3]

[4, 0, 5]

[7, 8, 6]

[1, 2, 3]

[4, 5, 0]

[7, 8, 6]

[1, 2, 3]

[4, 5, 6]

[7, 8, 0]