LAB - 2

18/10/24

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
```

classmate

Date

Page

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 cleaned

Location A already clean

Goal 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

else generate all successors of Node and

add generated nodes to the back of fringe

End loop

Consider initial and final state

$$\underset{\text{initial}}{\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 0 & 7 & 8 \end{matrix}} \longleftrightarrow \underset{\text{final}}{\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{matrix}}$$

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 0 & 7 & 8 \end{matrix}$$

$$\begin{matrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 4 & 7 & 8 \end{matrix} \qquad \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 0 & 8 \end{matrix}$$

$$\begin{matrix} 0 & 2 & 3 \\ 1 & 5 & 6 \\ 4 & 7 & 8 \end{matrix} \qquad \begin{matrix} 1 & 2 & 3 \\ 5 & 0 & 6 \\ 4 & 7 & 8 \end{matrix} \qquad \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{matrix} \qquad \begin{matrix} 1 & 2 & 3 \\ 4 & 0 & 6 \\ 7 & 5 & 8 \end{matrix}$$

final state

Date
Page

a) Implement 8 puzzle problem using DFS algorithm

ans) <u>Algorithm</u>:

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 Ok
4 7 8

---

2 0 3
1 5 6
4 7 8

⋮

1 2 3
4 5 6
7 8 0

final state

```python
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.")
```

```python
            else:
                print("Location B is already clean.")
        else:
            print("Location A is already clean.")

            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.")

    else:  # Vacuum is placed in Location B
        print("Vacuum is placed in Location B")

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

            if status_input_complement == '1':  # If A is Dirty
                print("Location A is Dirty.")
                print("Moving left to Location A.")
                cost += 1  # Cost for moving left
                print("Cost for moving LEFT: " + str(cost))
```

```python
                print("Cost for moving LEFT: " + str(cost))
                # Clean A
                goal_state['A'] = '0'
                cost += 1  # Cost for cleaning
                print("Cost for CLEANING A: " + str(cost))
                print("Location A has been Cleaned.")
            else:
                print("Location A is already clean.")
        else:
            print("Location B is already clean.")

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

    # Done cleaning
    print("GOAL STATE: ")
    print(goal_state)
    print("Performance Measurement: " + str(cost))

print("Tanish M V")
print("1BM22CS302")
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
```

```python
#8 puzzle using DFS
class PuzzleState:
    def __init__(self, state, parent=None):
        self.state = state
        self.parent = parent

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

    def get_possible_moves(self):
        moves = []
        zero_pos = self.state.index(0)

        directions = [
            (-3, "Up"),
            (3, "Down"),
            (-1, "Left"),
            (1, "Right")
        ]

        for direction, move in directions:
            new_pos = zero_pos + direction
            if 0 <= new_pos < 9:
                if (move == "Left" and zero_pos % 3 == 0) or (move == "Right" and new_pos % 3 == 0):
                    continue
                new_state = self.state[:]
                new_state[zero_pos], new_state[new_pos] = new_state[new_pos], new_state[zero_pos]
                moves.append(new_state)

        return moves

    def is_goal_state(self):
        return self.state == [1, 2, 3, 4, 5, 6, 7, 8, 0]


def dfs(initial_state, goal_state):
    stack = [PuzzleState(initial_state)]
```

```
    visited.add(tuple(initial_state))

    while stack:
        current_state = stack.pop()

        if current_state.is_goal_state():
            solution = []
            while current_state:
                solution.append(current_state.state)
                current_state = current_state.parent
            solution.reverse()
            return solution

        for next_state in current_state.get_possible_moves():
            if tuple(next_state) not in visited:
                visited.add(tuple(next_state))
                stack.append(PuzzleState(next_state, current_state))

    return None


def print_solution(solution):
    if solution:
        print("Solution:")
        for state in solution:
            print("\n".join([str(state[i:i+3]) for i in range(0, 9, 3)]))
            print()
    else:
        print("No solution found.")


initial_state = [1, 2, 3, 4, 0, 5, 7, 8, 6]
goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]

```

```python


initial_state = [1, 2, 3, 4, 0, 5, 7, 8, 6]
goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]

print("Tanish M V")
print("1BM22CS302")
print("8 puzzle using DFS:")

solution = dfs(initial_state, goal_state)

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]
```

```python
#8 puzzle using bfs
from collections import deque

class PuzzleState:
    def __init__(self, state, parent=None):
        self.state = state
        self.parent = parent

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

    def get_possible_moves(self):
        moves = []
        zero_pos = self.state.index(0)

        directions = [
            (-3, "Up"),
            (3, "Down"),
            (-1, "Left"),
            (1, "Right")
        ]

        for direction, move in directions:
            new_pos = zero_pos + direction
            if 0 <= new_pos < 9:
                if (move == "Left" and zero_pos % 3 == 0) or (move == "Right" and new_pos % 3 == 0):
                    continue
                new_state = self.state[:]
                new_state[zero_pos], new_state[new_pos] = new_state[new_pos], new_state[zero_pos]
                moves.append(new_state)

        return moves

```

```python
        return moves

    def is_goal_state(self):
        return self.state == [1, 2, 3, 4, 5, 6, 7, 8, 0]


def bfs(initial_state, goal_state):
    queue = deque([PuzzleState(initial_state)])
    visited = set()
    visited.add(tuple(initial_state))

    while queue:
        current_state = queue.popleft()

        if current_state.is_goal_state():
            solution = []
            while current_state:
                solution.append(current_state.state)
                current_state = current_state.parent
            solution.reverse()
            return solution

        for next_state in current_state.get_possible_moves():
            if tuple(next_state) not in visited:
                visited.add(tuple(next_state))
                queue.append(PuzzleState(next_state, current_state))

    return None


def print_solution(solution):
    if solution:
```

```python
def print_solution(solution):
    if solution:
        print("Solution Path (from initial to goal):")
        for state in solution:
            print("\n".join([str(state[i:i+3]) for i in range(0, 9, 3)]))
            print()
    else:
        print("No solution found.")


initial_state = [1, 2, 3, 4, 0, 5, 7, 8, 6]
goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]

print("Tanish M V")
print("1BM22CS302")
print("8 puzzle using BFS:")

solution = bfs(initial_state, goal_state)

print_solution(solution)

```

```
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]
```