Bafna Gold
Date: Page:

4/11/23

Lab - 4

Write an algorithm for 8-puzzle problem:-

Algorithm:-

1. Start with an empty queue
   queue → []

2. Enque with initial state.
   queue.append(src)

3. while the queue is not empty, dequeue the 1st state from the queue
   def solve_puzzle (initial state, goal-state)
   queue = deque ([initial state, []])

4. Check if the current state is the target state., if it is return "success".
   def check_current: (initial_state, goal_state)
   if (initial_state == goal_state)
   print ("success");

5. Generate all possible moves from current state by determining the direction of the empty spot & considering all possible direction.
   def generate (current_state):

6. moves = []
   blank_row, blank_column = find_blank(board
   possible_moves = [(1,0),(-1,0),(0,1),(0,-1)]

6. For each possible moves, calculate the resulting st

7. Check if the resulting state have already been visited. If it has not, enqueue the resulting state.

8. If the target state has not been found, continue with the next iteration with loop.

9. ~~If the target is not found after visiting all possible states return -1.~~

Found

⇒ Code:-

```
from collection import deque.

def find_blank(board):
    for i in range(3):
        for j in range(3):
            if board[i][j] == 0:
                return i, j
```

```
def generate_moves (board):
    moves = []
    blank_row, blank_col = find_blank(board)

    possible_moves = [
                    (1,0), (-1,0), (0,1), (0,-1)
    ]

for dr, dc in possible_moves:
    new_row, new_col = blank_row + dr, blank_col + dc

if 0 <= new_row < 3 and 0 <= new_col < 3:
    new_board = [row[:] for row in board]
    new_board[blank_row][blank_col], new_board[new_row][new_col] =
    new_board[new_row][new_col], new_board[blank_row][blank_col]
    moves.append(new_board)

return moves

def sol_puzzle(initial_state, goal_state):
    visited = set()
    queue = deque([(initial_state, [])])
while queue:
    current_state, path = queue.popleft()
    visited.add(tuple(map(tuple, current_state)))

    if current_state == goal_state:
        return path

possible_moves = generate_moves(current_state)
```

```
def print_steps(solution_path):
    if solution_path:
        print("Steps to reach the goal:")
        for step in solution_path:
            print("- - - - - -")
            for row in step:
                print("|", end=" ")
                for val in row:
                    if val == 0:
                        print(" ", end="| ")
                    else:
                        print(val, end="| ")
                print()
            print("- - - - - -")
            print()
```

```
initial = [
    [ , 2, 3]
    [4, 0, 5]
    [6, 7, 8]
]
goal = [
    [0, 1, 2]
    [3, 4, 5]
    [1, 7, 8]
]
```

```
Solution_path = solve_puzzle(initial, goal)
print_steps(solution_path)
```

Output:

Steps to reach the goal:

1 2 3
4 5
6 7 8

1 2 5
3 4
6 7 8

2 3
1 4 5
6 7 8

1 2
3 4 5
6 7 8

2 3
1 4 5
6 7 8

1 2
3 4 5
6 7 8

2 3 5
1 4
6 7 8

1 2
3 4 5
6 7 8

2 5
1 3 4
6 7 8

2 5
1 3 4
6 7 8

1 2 5
3 4
6 7 8

```
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 | 0 | 6
7 | 5 | 8
-----------
1 | 2 | 3
4 | 5 | 6
7 | 8 | 0
-----------
Success
```