

08/10/2024

## LAB 02- 8 PUZZLES PROBLEM

### ALGORITHM/PSEUDOCODE-

classmate  
Date 08/10/24  
Page 03

WEEK-2

Solve 8 puzzles problems, implement iterative deepening search algorithm

PSEUDOCODE:

```
FUNCTION iddfs(start-state)
    depth = 0
    While true
        result = depth-limited-search(start-state, depth)
        if result is not NONE then
            return result
        depth = depth + 1

FUNCTION depth-limited-search(state, limit)
    if state is the goal then
        return state
    if limit == 0 then
        return none

    for each next-state in get-next-states(state) do
        result = depth-limited-search(next-state, limit-1)
        if result is not NONE then
            return result
    return none

FUNCTION get-next-states(state)
    blank-position = find-blank(state)
    next-states = EMPTY-LIST

    for each move IN moves DO
        new-position = blank-position + move
        if new-position is valid then
```

```

blank_position = find_blank(state)
next_states = EMPTY_LIST

for each move IN moves DO
    new_position = blank_position + move
    if new_position is valid then
        new_state = DEEP_COPY(state)
        SWAP blank tile with adjacent tile in new_state
    return next_states

FUNCTION find_blank(state)
    for each row in state do
        for each tile in row do
            if tile == 0 then
                return row_index, col_index

FUNCTION is_goal(state)
    return state == goal_state

if solution is not NONE then
    print "solution found"
    print solution
else
    print "No solution found"

```

## CODE/INPUT-

```

import copy
# Define the goal state (3x3 board)
goal_state = [[1, 2, 3],
               [4, 5, 6],
               [7, 8, 0]]

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

def is_goal(state):
    return state == goal_state

```

```

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

def get_next_states(state):
    blank_x, blank_y = find_blank(state)
    next_states = []

    for move in moves:
        new_x, new_y = blank_x + move[0], blank_y + move[1]

        if 0 <= new_x < 3 and 0 <= new_y < 3:
            new_state = copy.deepcopy(state)
            new_state[blank_x][blank_y], new_state[new_x][new_y] =
new_state[new_x][new_y], new_state[blank_x][blank_y]
            next_states.append(new_state)

    return next_states

def iddfs(start_state):
    depth = 0
    while True:
        result = depth_limited_search(start_state, depth)
        if result:
            return result
        depth += 1

def depth_limited_search(state, limit):
    if is_goal(state):
        return state
    if limit == 0:
        return None

    for next_state in get_next_states(state):
        result = depth_limited_search(next_state, limit - 1)

```

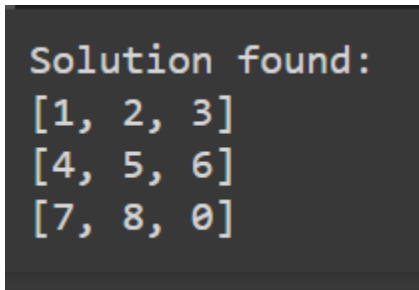
```
        if result:
            return result
    return None

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

solution = iddfs(initial_state)

if solution:
    print("Solution found:")
    for row in solution:
        print(row)
else:
    print("No solution found.")
```

## OUTPUT-



```
Solution found:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```