Bafna Gold
Date: Page:

8 Puzzle problem using DFS

Algorithm:

Step 1 start:

step 2. Assign initial state and goal states:

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

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

step3: Calc - manhantlan distance

initial_pos = x_1, y_1

goal_pos = x_2, y_2

```
for i in range(3):
    for j in ran(3):
        title = self.state[i][j]
        if title != 0:
            total_dist = (x2 - x1) + (y2 - y1)
```

Step 4: Get possible moves.

swap up, down, left, right

first find the position of empty space and return the position.

```
for dx, dy in directions:
    new_x, new_y = empty_x + dx, empty_y + dy
    if 0 <= new_x < 3 and 0 <= new_y < 3:

    return rows.
```

Step 5: dfs_with_manhantan function.

Implement DFS approach using a stack.

```
while stack:
    node := stack.pop()

if node.state == goal_state:
        return construct_solution(node)

visited.add(tuple(map(tuple, node.state)))
return None
```

Bafna Gold
Date: Page:

Step 6 Repeate the find_empty_space function and until the goal state is reached.

Step 7 Construct solution function.

Backtrack from the goal node to the start node

Proceed

Code:

```
class Node:
    def __init__(self, state, parent = None,
                 move = None, depth = 0):
        self.state = state
        self.parent = parent
        self.move = move
        self.depth = depth
        self.manhattan_distance = self.calculate_
                                  m_dist()

def cal_man_dist(self):
    total_dist = 0

    go_pos = { 1:(0,0), 2:(0,1), 3:(0,2),
               5:(1,0), 8:(1,1), 6:(1,2),
               0:(2,0), 7:(2,1), 4:(2,2)
             }
```

```
for i in range(3):
    for j in range(3):
        tile = self.s[i][j]
        if t != 0:
            x2, y2 = g[t]
            t_dist += abs(x2-i) + abs(y-j)
return t_distan
```

```
def ~~is_sol~~ dfs_with_man(initial_st, g_st):

stack = [N(in_st)]
visited = set()

while stack:
    node = stack.pop()

    if node.state == goal_state:
        return const_sol(node)

visited.add(tuple(map(tuple, node.stat)))

for new_s, move in g_p_m(node.stat):
    if tuple(map(tuple, new_st)) not in
        visited:

new_node = Node(n_st, parent = n, move =

if new_node.manhattan <= node.manhatt
    stack.append(new_node):
return None
```

```
def find_empty_space(s)
    for i in range(3):
        for j in range(3):
            if s[i][j] == 0:
                return i, j
    return None

def construct_soln(n):
    path = c
    while node.parent is not None:
        path.append(n)
        node = node.parent
    path.reverse()

for step in path:
    print_puzzle(step.state)
    print()
```

output:

solution found!!

8/10/24