classmate
Date 8/10/24
Page 10

LAB-3 8-puzzle

Algorithm for DFS

1. Initial state.
   - Represent the initial in a list or 2D array
   - Calculate Manhattan Distance. (sum of the differences b/w the tile position ~~wrong~~ to the target position)
   - Push the initial state onto stack.

2. DFS Search
   while the stack is ~~not~~ empty
   - Pop current node
   - if popped element is the goal state, match and terminate.
   - if the element is not visited then add it to the visited set
   - Generate all possible moves (left, right down, up) based on empty posn.
   - for valid move.
     - Compute the resulting state.
     - ~~Calculate manhattan Distance~~
     - If new state is not visited push to stack.

3. Move
   - for a valid move, swap the blank tile with the one near and create a new state

Algorithm for Manhattan Distance

4. Calculate Manhattan Distance
   - calculate the distance for all tiles except the blank tile.
   - for i in position (x, y), and goal position is (x_1, y_1)
     $|x - x_1| + |y - y_1|$
   - sum of the differences b/w the tile to target position

5. ~~Repeat the above, till the puzzle is solved.~~

1. Initial state.
   - Represent the initial state in a list of 2D array
   - Calculate Manhattan Distance.
   - Insert the initial state to the queue Priority. if equal to the Manhattan Distance.

2. Search:
   While queue not empty.
   - pop lowest value from queue.
   - if popped element is the target, match and return it.
   - if state not visited most ~~add~~ to visited set
   - Generate all valid moves
     for each valid move
     - calculate a new state.
     - calculate Manhattan distance.

3. Move
   - for a valid move, swap the blank tile with one near and create a new state.

DFS

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

Manhattan.

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

goal state.

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

Code:- DFS

```
Class PS:
    def __init__(self, board, ep, moves=[]):
        self.board = board
        self.ep = ep
        self.moves = moves
def goal(self):
    return self.board == [1, 2, 3, 4, 5, 6, 7, 8, 0]
```

```
def possmoves(self):
    x, y = self.ep
    move = []
    for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
        nx, ny = x+dx, y+dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            nboard = self.board[:]
            nboard[x*3+y], nboard[nx*3+ny] =
            nboard[nx*3+ny], nboard[x*3+y]
            moves.append((nboard, (nx, ny)))
    return moves

def dfs(is):
    stack, visited = [is], set()
    while stack:
        cs = stack.pop()
        if cs.goal():
            return cs.move
        visited.add(tuple(cs.board))
        for nboard, nep in cs.possmoves():
            n.state = puzzlestate(nboard, nep, cs.move+
                [nboard])
            if tuple(nboard) not in visited:
                stack.append(nstate)
    return None

def pm(board):
    for i in range(0,9,3):
        print(board[i:i+3])
    print()
```

classmate

Date
Page

```
def main():
    ib=[1,2,3,4,0,5,7,8,6]
    ep= id.index(0)
    is =PuzzleState(ib,(ep//3, ep%3))

    print("Initial State:")
    print_matrix(ib)
    solution=dfs(is)
    if Solution:
        print("Solution found:"
        for step in solution:
            print_matrix(step)
    else:
        print("No Solution found.")
if __name__ =="__main__":
    main()
```

O/P :-

Initial state

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

Solution found:

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

8/10/24

classmate
Date
Page

Code - Manhattan :-

```
def md(state, gs):
    dis = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                goal_i = (state[i][j] - 1) // 3
                goal_j = (state[i][j] - 1) % 3
                distance = abs(i - goal_i) + abs(j - goal_j)
    return dis
def getneig(state):
    i, j = next((i, j) for i in range(3) for j in range(3)
        if state[i][j] == 0)
    moves = [(i-1, j), (i+1, j), (i, j-1), (i, j+1)]
    return [swap(state, i, j, x, y) for x, y in moves
    if 0 <= x < 3 and 0 <= y < 3]
def swap(state, i1, j1, i2, j2):
    nstate = [row[:] for row in state]
    nstate[i1][j1], nstate[i2][j2] = nstate[i2][j2],
                                     nstate[i1][j1]
    return nstate.
def dfsm(start, goal, visited = set()):
    if state == goal: return [state]
    visited.add(str(state))
    neig = sorted(getneig(state), key=lambda x: md(x, goal))
    for neighbor in neig:
        if str(neighbor) not in visited:
            path = dfsm(neighbor, goal, visited)
            if path: return [state] + path
```

return None.

```
is = [[int(x) for x in input("Enter row {i+1}: ").split()]
      for i in range(3)]
gs = [[1,2,3],[4,5,6],[7,8,0]]
sol = dfsm(is, gs)
if sol:
    print("Solution found:")
    for state in sol: print(*state, sep='\n', end='\n')
else:
    print("No solution found")
```

O/P

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