8/10/24

DATE:

PAGE:

Lab-2

8 - Puzzle Problem

Pseudo Code

```
Class Node:
    Function - init(state, parent), action, path_cost
        Set self.state = state
        Set self.parent = parent
        Set self.action = action
        Set self.path_cost = path_cost
    function expand()
        Create children
        Set row, col = find_blank()
        Create possible_actions
        if row > 0 then add 'up' to possible_actions
        if row < 2 then add 'down' to possible_actions
        if col > 0 then add 'left' to possible_actions
        if col < 2 then add 'right' to possible_actions
        for action in possible_actions:
            Create new_state as a copy of
            self.state
            if action == 'up' then swap new_state
            [row][col] with new_state[row-1]
            else if action == "down" then swap
            new_state[row][col] with new_state
            [row+1][col]
            else if action == "left" then swap
            new_state[row][col] with new_state
            [row][col-1]
            else if action == "right" then swap
            new_state[row][col] with new_state
            [row][col+1]
```

DATE: PAGE:

```
append newnode(new_state, self, action,
  self.path_cost to Children
  return children
function find_blank():
    for row from 0 to 2:
      for col in from 0 to 2:
        if self.state[row][col] == 0 then
          return (row)[col]
function depth_first_search(initial state, goal state
  Set frontier = [Node(initial_state)]
  Set explored = empty set
  While frontier is not empty:
  Set node = frontier.pop()
  if node.state == goal_state then
    return node.
  add tuple of node state to explored:
  for child in node.expand():
    if tuple of child state not in explored
      then append child to frontier
    return none
function print_solution(node):
      create path
      while node is not none:
      append(node.action, node.state) to path
      self.node = node.parent
        reverse path
      for (action, state) in path:
      if action is not none then print
      "action": action
        print state
        print("")
```

DATE: PAGE:

set initial_state = [[1, 2, 3], [0, 4, 6], [7, 5, 8]]
set goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
set solution, depth = bfs_search(initial_state, goal_state)

if solution is not none then:
    print "solution found"
    call print_solution(solution)
else:
    print "solution not found"