**LAB 4:**

**Program:** A\* Algorithm with Misplaced Tiles Heuristic

Algorithm: CLASS PuzzleState:

FUNCTION \_\_init\_\_(board, zero\_pos, moves=0):

SET self.board = board

SET self.zero\_pos = zero\_pos

SET self.moves = moves

SET self.heuristic = self.calculate\_misplaced\_tiles()

FUNCTION calculate\_misplaced\_tiles():

SET goal\_state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]

RETURN SUM(1 FOR i IN range(3) FOR j IN range(3) IF self.board[i][j] != goal\_state[i][j])

FUNCTION get\_neighbors():

SET neighbors = []

SET x, y = self.zero\_pos

SET directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

FOR (dx, dy) IN directions:

SET new\_x = x + dx

SET new\_y = y + dy

IF (0 <= new\_x < 3 AND 0 <= new\_y < 3):

SET new\_board = COPY of self.board

SWAP new\_board[x][y] WITH new\_board[new\_x][new\_y]

APPEND (new\_board, (new\_x, new\_y)) TO neighbors

RETURN neighbors

FUNCTION \_\_lt\_\_(other):

RETURN (self.moves + self.heuristic) < (other.moves + other.heuristic)

FUNCTION \_\_repr\_\_():

RETURN STRING representation of self.board with moves and heuristic

FUNCTION a\_star(initial\_board):

SET initial\_zero\_pos = FIND position (i, j) WHERE initial\_board[i][j] == 0

SET initial\_state = PuzzleState(initial\_board, initial\_zero\_pos)

INITIALIZE priority\_queue as empty

HEAP\_PUSH priority\_queue WITH initial\_state

INITIALIZE visited as empty set

WHILE priority\_queue is not empty:

SET current\_state = HEAP\_POP priority\_queue

PRINT "Exploring State:", current\_state

IF current\_state.heuristic == 0:

PRINT "Reached goal in", current\_state.moves, "moves."

RETURN

ADD tuple representation of current\_state.board TO visited

FOR (neighbor\_board, neighbor\_zero\_pos) IN current\_state.get\_neighbors():

SET neighbor\_state = PuzzleState(neighbor\_board, neighbor\_zero\_pos, current\_state.moves + 1)

IF tuple representation of neighbor\_board NOT IN visited:

HEAP\_PUSH priority\_queue WITH neighbor\_state

PRINT "No solution found."

FUNCTION main():

SET initial\_board = [

[2, 8, 3],

[1, 6, 4],

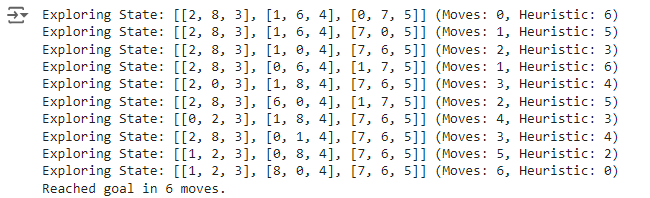
[0, 7, 5]

]

CALL a\_star(initial\_board)

CALL main()

**Output:**

****

### Program: A\* Search in 8-Puzzle using Manhattan

### Algorithm:

### CLASS PuzzleState:

### FUNCTION \_\_init\_\_(board, zero\_pos, moves=0):

### SET self.board = board

### SET self.zero\_pos = zero\_pos

### SET self.moves = moves

### SET self.heuristic = self.calculate\_manhattan\_distance()

### FUNCTION calculate\_manhattan\_distance():

### SET distance = 0

### SET goal\_positions = {

### 1: (0, 0), 2: (0, 1), 3: (0, 2),

### 8: (1, 0), 0: (1, 1), 4: (1, 2),

### 7: (2, 0), 6: (2, 1), 5: (2, 2)

### }

### FOR i IN range(3):

### FOR j IN range(3):

### SET tile = self.board[i][j]

### IF tile != 0:

### SET goal\_x, goal\_y = goal\_positions[tile]

### distance += abs(i - goal\_x) + abs(j - goal\_y)

### RETURN distance

### FUNCTION get\_neighbors():

### SET neighbors = []

### SET x, y = self.zero\_pos

### SET directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

### FOR (dx, dy) IN directions:

### SET new\_x = x + dx

### SET new\_y = y + dy

### IF (0 <= new\_x < 3 AND 0 <= new\_y < 3):

### SET new\_board = COPY of self.board

### SWAP new\_board[x][y] WITH new\_board[new\_x][new\_y]

### APPEND (new\_board, (new\_x, new\_y)) TO neighbors

### RETURN neighbors

### FUNCTION \_\_lt\_\_(other):

### RETURN (self.moves + self.heuristic) < (other.moves + other.heuristic)

### FUNCTION \_\_repr\_\_():

### RETURN STRING representation of self.board with moves and heuristic

### FUNCTION a\_star(initial\_board):

### SET initial\_zero\_pos = FIND position (i, j) WHERE initial\_board[i][j] == 0

### SET initial\_state = PuzzleState(initial\_board, initial\_zero\_pos)

### INITIALIZE priority\_queue as empty

### HEAP\_PUSH priority\_queue WITH initial\_state

### INITIALIZE visited as empty set

### WHILE priority\_queue is not empty:

### SET current\_state = HEAP\_POP priority\_queue

### PRINT "Exploring State:", current\_state

### IF current\_state.heuristic == 0:

### PRINT "Reached goal in", current\_state.moves, "moves."

### RETURN

### ADD tuple representation of current\_state.board TO visited

### FOR (neighbor\_board, neighbor\_zero\_pos) IN current\_state.get\_neighbors():

### SET neighbor\_state = PuzzleState(neighbor\_board, neighbor\_zero\_pos, current\_state.moves + 1)

### IF tuple representation of neighbor\_board NOT IN visited:

### HEAP\_PUSH priority\_queue WITH neighbor\_state

### PRINT "No solution found."

### FUNCTION main():

### SET initial\_board = [

### [2, 8, 3],

### [1, 6, 4],

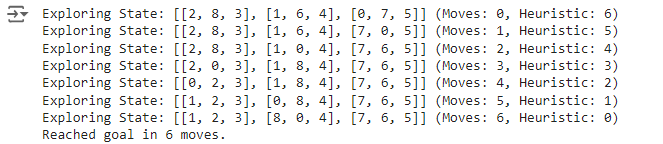
### [0, 7, 5]

### ]

### CALL a\_star(initial\_board)

### CALL main()

**Output:**

****