Assignment No 01: 8-Puzzle Problem

CSE-0408 Summer 2021

Sheikh Afrin
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
sheikhafrin2016@gmail.com

***Abstract*—The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state. Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile. The empty space cannot move diagonally and can take only one step at a time.**

***Index Terms*—Python**

I. Introduction

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell.
The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.

II. Literature Review

Sadikov and Bratko (2006) studied the suitability of pessimistic and optimistic heuristic functions for a real-time search in the 8-puzzle. They discovered that pessimistic functions are more suitable. They also observed the pathology, which was stronger with the pessimistic heuristic function. However, they did not study the influence of other factors on the pathology or provide any analysis of the gain of a deeper search. In our paper, the basic pathology observed in (Sadikov and Bratko 2006) was confirmed.

III. Proposed Methodology

The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.

IV. Rules for solving the puzzle

Instead of moving the tiles in the empty space, we can visualize moving the empty space in place of the tile, basically swapping the tile with the empty space. The empty space can only move in four directions.

1.Up
2.Down
3. Right or
4. Left

V. CODE

```
In [1]: from copy import deepcopy
        from colorama import Fore, Back, Style

        DIRECTIONS = {"D": [-1, 0], "U": [1, 0], "R": [0, -1], "L": [0, 1]}
        END = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]

        # unicode
        left_down_angle = '\u2514'
        right_down_angle = '\u2518'
        right_up_angle = '\u2510'
        left_up_angle = '\u250C'

        middle_junction = '\u253C'
        top_junction = '\u252C'
        bottom_junction = '\u2534'
        right_junction = '\u2524'
        left_junction = '\u251C'

        bar = Style.BRIGHT + Fore.CYAN + '\u2502' + Fore.RESET + Style.RESET_ALL
        dash = '\u2500'

        first_line = Style.BRIGHT + Fore.CYAN + left_up_angle + dash + dash + dash + top_junction + dash + dash + dash + top_junction + d
        middle_line = Style.BRIGHT + Fore.CYAN + left_junction + dash + dash + dash + middle_junction + dash + dash + dash + middle_junct
        last_line = Style.BRIGHT + Fore.CYAN + left_down_angle + dash + dash + dash + bottom_junction + dash + dash + dash + bottom_junct


        def print_puzzle(array):
            print(first_line)
            for a in range(len(array)):
```

Fig. 1.

```
def print_puzzle(array):
    print(first_line)
    for a in range(len(array)):
        for i in array[a]:
            if i == 0:
                print(bar, Back.RED + ' ' + Back.RESET, end=' ')
            else:
                print(bar, i, end=' ')
        print(bar)
        if a == 2:
            print(last_line)
        else:
            print(middle_line)

class Node:
    def __init__(self, current_node, previous_node, g, h, dir):
        self.current_node = current_node
        self.previous_node = previous_node
        self.g = g
        self.h = h
        self.dir = dir

    def f(self):
        return self.g + self.h


def get_pos(current_state, element):
    for row in range(len(current_state)):
        if element in current_state[row]:
            return (row, current_state[row].index(element))
```

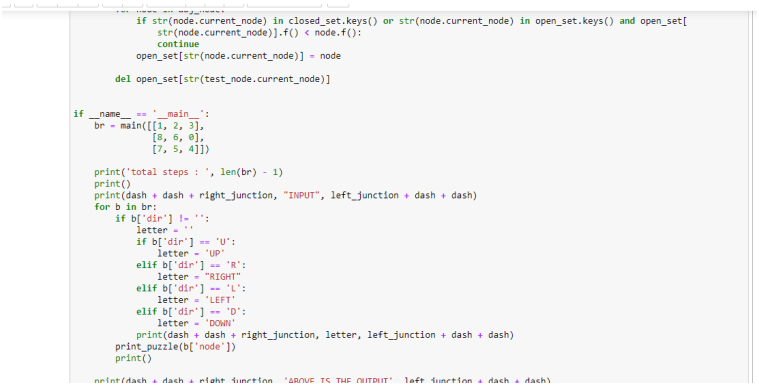
Fig. 2.

```
def euclidianCost(current_state):
    cost = 0
    for row in range(len(current_state)):
        for col in range(len(current_state[0])):
            pos = get_pos(END, current_state[row][col])
            cost += abs(row - pos[0]) + abs(col - pos[1])
    return cost

def getAdjNode(node):
    listNode = []
    emptyPos = get_pos(node.current_node, 0)

    for dir in DIRECTIONS.keys():
        newPos = (emptyPos[0] + DIRECTIONS[dir][0], emptyPos[1] + DIRECTIONS[dir][1])
        if 0 <= newPos[0] < len(node.current_node) and 0 <= newPos[1] < len(node.current_node[0]):
            newState = deepcopy(node.current_node)
            newState[emptyPos[0]][emptyPos[1]] = node.current_node[newPos[0]][newPos[1]]
            newState[newPos[0]][newPos[1]] = 0
            # listNode += [Node(newState, node.current_node, node.g + 1, euclidianCost(newState), dir)]
            listNode.append(Node(newState, node.current_node, node.g + 1, euclidianCost(newState), dir))

    return listNode

def getBestNode(openSet):
    firstIter = True

    for node in openSet.values():
        if firstIter or node.f() < bestF:
            firstIter = False
            bestNode = node
            bestF = bestNode.f()
    return bestNode
```

Fig. 3.

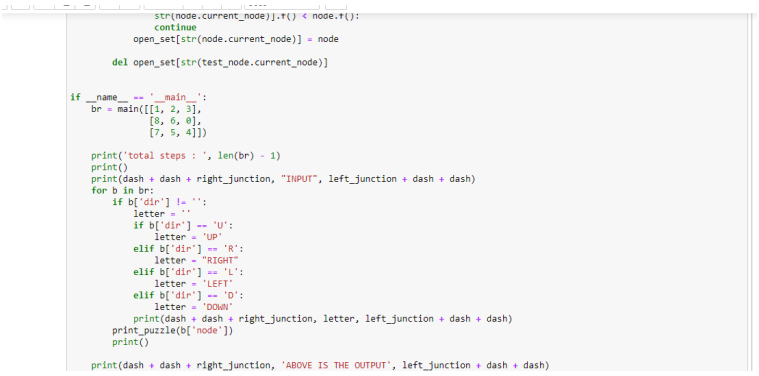
```
def buildPath(closedSet):
    node = closedSet[str(END)]
    branch = list()

    while node.dir:
        branch.append({
            'dir': node.dir,
            'node': node.current_node
        })
        node = closedSet[str(node.previous_node)]
    branch.append({
        'dir': '',
        'node': node.current_node
    })
    branch.reverse()

    return branch

def main(puzzle):
    open_set = {str(puzzle): Node(puzzle, puzzle, 0, euclidianCost(puzzle), "")}
    closed_set = {}

    while True:
        test_node = getBestNode(open_set)
        closed_set[str(test_node.current_node)] = test_node

        if test_node.current_node == END:
            return buildPath(closed_set)

        adj_node = getAdjNode(test_node)
        for node in adj_node:
```

Fig. 4.



```
            if str(node.current_node) in closed_set.keys() or str(node.current_node) in open_set.keys() and open_set[
                str(node.current_node)].f() < node.f():
                continue
            open_set[str(node.current_node)] = node

        del open_set[str(test_node.current_node)]


if __name__ == '__main__':
    br = main([[1, 2, 3],
               [8, 6, 0],
               [7, 5, 4]])

    print('total steps : ', len(br) - 1)
    print()
    print(dash + dash + right_junction, "INPUT", left_junction + dash + dash)
    for b in br:
        if b['dir'] != '':
            letter = ''
            if b['dir'] == 'U':
                letter = 'UP'
            elif b['dir'] == 'R':
                letter = "RIGHT"
            elif b['dir'] == 'L':
                letter = 'LEFT'
            elif b['dir'] == 'D':
                letter = 'DOWN'
            print(dash + dash + right_junction, letter, left_junction + dash + dash)
        print_puzzle(b['node'])
        print()

    print(dash + dash + right_junction, 'ABOVE IS THE OUTPUT', left_junction + dash + dash)
```

Fig. 5.



```
            str(node.current_node)].f() < node.f():
            continue
        open_set[str(node.current_node)] = node

    del open_set[str(test_node.current_node)]


if __name__ == '__main__':
    br = main([[1, 2, 3],
               [8, 6, 0],
               [7, 5, 4]])

    print('total steps : ', len(br) - 1)
    print()
    print(dash + dash + right_junction, "INPUT", left_junction + dash + dash)
    for b in br:
        if b['dir'] != '':
            letter = ''
            if b['dir'] == 'U':
                letter = 'UP'
            elif b['dir'] == 'R':
                letter = "RIGHT"
            elif b['dir'] == 'L':
                letter = 'LEFT'
            elif b['dir'] == 'D':
                letter = 'DOWN'
            print(dash + dash + right_junction, letter, left_junction + dash + dash)
        print_puzzle(b['node'])
        print()

    print(dash + dash + right_junction, 'ABOVE IS THE OUTPUT', left_junction + dash + dash)
```

Fig. 6.

VI. Conclusion

I tested the code to see that how many state it would take to get from the current state to the goal state,I try many moves and it worked.

Acknowledgment

I would like to thank my honourable **Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] Piltaver, R., Luštrek, M., & Gams, M. (2012). The pathology of heuristic search in the 8-puzzle. Journal of Experimental & Theoretical Artificial Intelligence, 24(1), 65-94.

[2] Nayak, D. (2014). Analysis and Implementation of Admissible Heuristics in 8 Puzzle Problem (Doctoral dissertation).

[3] Gaschnig, J. (1981). A problem similarity approach to devising heuristics: First results. In Readings in Artificial Intelligence (pp. 23-29). Morgan Kaufmann.