

8 puzzle problem using A\* algorithm

class Node:

    def \_\_init\_\_(self, data, level, fval):

        self.data = data

        self.level = level

        self.fval = fval

    def generate\_children(self):

        x, y = self.find\_blank()

        directions = [(x, y-1), (x, y+1), (x-1, y), (x+1, y)]

        children = []

        for new\_x, new\_y in directions:

            child\_data = self.move\_blank(x, y, new\_x, new\_y)

            if child\_data:

                children.append(Node(child\_data, self.level + 1, 0))

        return children

    def move\_blank(self, x1, y1, x2, y2):

        if 0 <= x2 < len(self.data) and 0 <= y2 < len(self.data):

            new\_data = [row[:] for row in self.data]

            new\_data[x1][y1], new\_data[x2][y2] = new\_data[x2][y2], new\_data[x1][y1]

            return new\_data

        return None

    def find\_blank(self):

        for i, row in enumerate(self.data):

            for j, value in enumerate(row):

                if value == '\_':

                    return i, j

class Puzzle:

    def \_\_init\_\_(self, size):

        self.size = size

        self.open = []

        self.closed = []

    def input\_puzzle(self, prompt):

        print(prompt)

        return [input().split() for \_ in range(self.size)]

    def calculate\_f(self, start, goal):

        return self.calculate\_h(start.data, goal) + start.level

    def calculate\_h(self, start, goal):

        return sum(start[i][j] != goal[i][j] and start[i][j] != '\_' for i in range(self.size) for j in range(self.size))

    def solve(self):

        start = self.input\_puzzle("Enter the start state matrix:")

        goal = self.input\_puzzle("Enter the goal state matrix:")

        start\_node = Node(start, 0, 0)

        start\_node.fval = self.calculate\_f(start\_node, goal)

        self.open.append(start\_node)

        while self.open:

            current = self.open.pop(0)

            print("\n  | \n  | \n \\'/ \n")

            for row in current.data:

                print(" ".join(row))

            if self.calculate\_h(current.data, goal) == 0:

                break

            for child in current.generate\_children():

                child.fval = self.calculate\_f(child, goal)

                self.open.append(child)

            self.closed.append(current)

            self.open.sort(key=lambda node: node.fval)

if \_\_name\_\_ == "\_\_main\_\_":

    puzzle = Puzzle(3)

    puzzle.solve()

