**8 Puzzle**

**class Node:**

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

**self.data = data**

**self.level = level**

**self.fval = fval**

**def generate\_child(self):**

**x, y = self.find(self.data, '\_')**

**val\_list = [[x, y - 1], [x, y + 1], [x - 1, y], [x + 1, y]]**

**children = []**

**for i in val\_list:**

**child = self.shuffle(self.data, x, y, i[0], i[1])**

**if child is not None:**

**child\_node = Node(child, self.level + 1, 0)**

**children.append(child\_node)**

**return children**

**def shuffle(self, puz, x1, y1, x2, y2):**

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

**temp\_puz = []**

**temp\_puz = self.copy(puz)**

**temp = temp\_puz[x2][y2]**

**temp\_puz[x2][y2] = temp\_puz[x1][y1]**

**temp\_puz[x1][y1] = temp**

**return temp\_puz**

**else:**

**return None**

**def copy(self, root):**

**temp = []**

**for i in root:**

**t = []**

**for j in i:**

**t.append(j)**

**temp.append(t)**

**return temp**

**def find(self, puz, x):**

**for i in range(0, len(self.data)):**

**for j in range(0, len(self.data)):**

**if puz[i][j] == x:**

**return i, j**

**class Puzzle:**

**def \_\_init\_\_(self, size):**

**self.n = size**

**self.open = []**

**self.closed = []**

**def accept(self):**

**puz = []**

**for i in range(0, self.n):**

**temp = input().split(" ")**

**puz.append(temp)**

**return puz**

**def f(self, start, goal):**

**return self.h(start.data, goal) + start.level**

**def h(self, start, goal):**

**temp = 0**

**for i in range(0, self.n):**

**for j in range(0, self.n):**

**if start[i][j] != goal[i][j] and start[i][j] != '\_':**

**temp += 1**

**return temp**

**def process(self):**

**print("Enter the start state matrix \n")**

**start = self.accept()**

**print("Enter the goal state matrix \n")**

**goal = self.accept()**

**start = Node(start, 0, 0)**

**start.fval = self.f(start, goal)**

**self.open.append(start)**

**print("\n\n")**

**while True:**

**cur = self.open[0]**

**print("")**

**print(" | ")**

**print(" | ")**

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

**for i in cur.data:**

**for j in i:**

**print(j, end=" ")**

**print("")**

**if (self.h(cur.data, goal) == 0):**

**break**

**for i in cur.generate\_child():**

**i.fval = self.f(i, goal)**

**self.open.append(i)**

**self.closed.append(cur)**

**del self.open[0]**

**self.open.sort(key=lambda x: x.fval, reverse=False)**

**puz = Puzzle(3)**

**puz.process()**

**Enter the start state matrix**

**1 2 3**

**\_ 4 6**

**7 5 8**

**Enter the goal state matrix**

**1 2 3**

**4 5 6**

**7 8 \_**