Program 2: Solve 8 puzzle problem.

PRANAV TAGADEESH IBMI8CS071
PROGRAM - 2 Solve 8 puzzle problem
Given ia 3×3 board with 8 tiles (every tile han one
number from 1 to 8) and one empty space. The
objective is to place the numbers on tiles to match
Linal continuation weing the empty share. We can
final configuration using the empty space. We can slide four adjacent (left, right above and below)
tiles into the empty space.
pas
Program
lay Node:
Program Lan Node: def - init_ (self, data, level, fual): self, data = data self, fenel = level
self. data = data
self level = level
self, fual = fual
def generate_child (self):
x, y = relf - find (self - data, (_')
nal_lest = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
def generate_child (self): x, y = self - find (self - data, (_')) val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]] shildren = []
for in val-list:
cheld = self. struffle (self. data, x, y, "(o), i (1))
if child is not None:
for in val-list: child = self. shruffle (self. data, x, y, i [0], i [1]) if child is not None: child node = Node (child, self level +1,0) children append (child-rode)
heldren append (hild-rode)

```
return children
return children

def shuffle (self, forz, x1, y1, x2, y2):

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

and y2 < len (self.data):

temb-fuz = self.copy (fuz)

temb-fuz = self.copy (fuz)

temb-fuz = temp-fuz (x2) (y2)

temp-fuz (x2) (y2) = temp-fuz (x1) (y1)

temp-fuz [x1] (y1] = temp

seeturn temp-fuz

ohe:
class Puzzle
```



```
for i'm range (0, xelf.n):

temp = input () effect ("")

pux offend (temp)

orstwin pur
def f(selb; start, goal):
return self. h(start.data, goal) + start. level
def sh ( self , start , goal):
      for i in hange (0, self.n):

for j in hange (0, self.n):

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

temp += |

return temp
       return temps

def process(self):

prient ("Enter the start state matrix In")

start = self. weept ()

point ("Enter the goal state matrix In")

goal = self accept ()

start = Node (start, 0, 0)

start fral = Self. f(start, goal)

self. open.append (start)

prient ("In In")
                                        perint (" (")
```

	Date (
print (j, end = "") print ("") print ("")	
if (self . h (cur . dala) jane) -):
break for i in our generate = child(): i fral = self. f(i, goal) self open append (i) self dosed append (cur) del self open (0)	
self closed append (cur) del self open (0)	. // 0
del self open (0) self open sort (key = lamb da s eseverse = False)	e: x. fual;
puz = Puzzle (3) puz-process ()	
Output: Start State matrix 2 - 3	
1 8 4 7 6 5	
Goal state matrien	
8 - 4 7 6 5	
2 3	
1 8 4 7 6 5	
2 3	
7 6 5	

classmate

		classmate Date
1 2 3		
765		
 1 2 3		
8 - 4 7 6 5		

```
#8 puzzle problem
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 \ge 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
2 _ 3
1 8 4
7 6 5
Enter the goal state matrix
1 2 3
8 _ 4
7 6 5
7 6 5
_ 2 3
1 8 4
7 6 5
```

```
\'/
2 3
1 8 4
7 6 5
1 2 3
_ 8 4
7 6 5
1 2 3
8 _ 4
7 6 5
```