



Tecnológico de Monterrey - Campus Monterrey

School of Engineering and Sciences

Engineering in Computational Technologies

Analysis and Design of Advanced Algorithms

Homework 14: 8 Tree

Group: 607
Team #3

Luis Salomón Flores Ugalde

Santiago Quintana Moreno A01571222
Miguel Ángel Álvarez Hermida a01722925

File Edit Selection View Go Run Terminal Help

Q 2.Advanced Algorithms

A01571222_A01722925_8puzzle.py U

```
Homework14_Puzzle > A01571222_A01722925_8puzzle.py > Node > __init__
```

```
1 # Analysis and Design of Advanced Algorithms
2 # Group #607
3 # Team 3
4 # Luis Salomón Flores Ugalde
5
6 # Santiago Quintana Moreno A01571222
7 # Miguel Ángel Álvarez Hermida A01722925
8
9 # ----- HOMEWORK 14 PUZZLE SOLVER -----
10
11 import random
12 import itertools
13 import collections
14 import time
15
16 class Node:
17     """
18     A class representing an Solver node
19     - 'puzzle' is a Puzzle instance see that move() creates a new puzzle depending on the move
20     - 'parent' is the preceding node generated by the solver, if any
21     - 'action' is the action taken to produce puzzle, if any
22     """
23
24     def __init__(self, puzzle, parent=None, action=None):
25         self.puzzle = puzzle
26         self.parent = parent
27         self.action = action
28         if parent:
29             self.g = parent.g + 1
30         else:
31             self.g = 0
32
33
34     def state(self):
35         """
36         Return a hashable representation of self
37         """
38         return str(self)
39
40     def path(self):
41         """
42         Reconstruct path by walking back from current node to start
43         """
44         node, p = self, []
45         while node:
46             p.append(node)
47             node = node.parent
48         return p[1:-1]
```

(.jupyter) PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms> & "D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms.jupyter\Scripts\python.exe" "d:/1.SQM/1.UNIVERSIDAD/5. QUINTO SEMESTRE/2.Advanced Algorithms/Homework14_Puzzle/A01571222_A01722925_8puzzle.py"

None
[1, 2, 3]
[4, 5, 0]
[6, 7, 8]

U
[1, 2, 0]
[4, 5, 3]
[6, 7, 8]

L
[1, 0, 2]
[4, 5, 3]
[6, 7, 8]

D
[1, 5, 2]
[4, 0, 3]
[6, 7, 8]

L
[1, 5, 2]
[0, 4, 3]
[6, 7, 8]

U
[0, 5, 2]
[1, 4, 3]
[6, 7, 8]

R
[5, 0, 2]
[1, 4, 3]
[6, 7, 8]

R
[5, 2, 0]
[1, 4, 3]
[6, 7, 8]

D
[5, 2, 3]
[1, 4, 0]
[6, 7, 8]

D
[5, 2, 3]
[1, 4, 8]
[6, 7, 0]

L
[5, 2, 3]

File Edit Selection View Go Run Terminal Help

2.Advanced Algorithms

A01571222_A01722925_8puzzle.py U

```
Homework14_Puzzle > A01571222_A01722925_8puzzle.py > Node > __init__
```

```
16     class Node:
40         def path(self):
41             return p[::-1]
48
49         def solved(self):
50             """ Wrapper to check if 'puzzle' is solved """
51             return self.puzzle.solved()
52
53         def actions(self):
54             """ Wrapper for 'actions' accessible at current state """
55             return self.puzzle.actions()
56
57         def h(self):
58             """Calculate h value using Manhattan distance heuristic"""
59             return self.puzzle.manhattan()
60
61         def f(self):
62             """
63             Return f(n) = g(n) + h(n) for A*.
64             """
65             return self.g + self.h()
66
67         def __str__(self):
68             return str(self.puzzle)
69
70
71     class Solver:
72         """
73             '8-puzzle' solver
74             - 'start' is a Puzzle instance
75         """
76
77
78         def __init__(self, start):
79             self.start = start
80
81         def solve(self):
82             queue = collections.deque([Node(self.start)])
83             seen = set()
84             seen.add(queue[0].state())
85
86             while queue:
87                 queue = collections.deque(sorted(queue, key=lambda node: node.f()))
88                 node = queue.popleft()
89
90                 if node.solved():
91                     return node.path()
92
93                 for move, action in node.actions():
```

(.jupyter) PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms> & "D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms.jupyter\Scripts\python.exe" "d:/1.SQM/1.UNIVERSIDAD/5. QUINTO SEMESTRE/2.Advanced Algorithms/Homework14_Puzzle/A01571222_A01722925_8puzzle.py"

None
[1, 2, 3]
[4, 5, 0]
[6, 7, 8]

U
[1, 2, 0]
[4, 5, 3]
[6, 7, 8]

L
[1, 0, 2]
[4, 5, 3]
[6, 7, 8]

D
[1, 5, 2]
[4, 0, 3]
[6, 7, 8]

L
[1, 5, 2]
[0, 4, 3]
[6, 7, 8]

U
[0, 5, 2]
[1, 4, 3]
[6, 7, 8]

R
[5, 0, 2]
[1, 4, 3]
[6, 7, 8]

R
[5, 2, 0]
[1, 4, 3]
[6, 7, 8]

D
[5, 2, 3]
[1, 4, 0]
[6, 7, 8]

D
[5, 2, 3]
[1, 4, 8]
[6, 7, 0]

L
[5, 2, 3]

File Edit Selection View Go Run Terminal Help

AplicativoBodega/AplicativoApp#12 needs reviewers

Go Live

File Edit Selection View Go Run Terminal Help

2.Advanced Algorithms

A01571222_A01722925_8puzzle.py U

```
Homework14_Puzzle > A01571222_A01722925_8puzzle.py > Node > __init__
```

```
72     class Solver:
73         def solve(self):
74             for move, action in node.actions():
75                 child = Node(move(), node, action)
76
77                 if child.state() not in seen:
78                     queue.append(child)
79                     seen.add(child.state())
80
81
82     class Puzzle:
83         """
84             A class representing an '8-puzzle'.
85             - 'board' should be a square list of lists with integer entries 0...width^2 - 1
86             | e.g. [[1,2,3],[4,0,6],[7,5,8]]
87             - 'goal_state' is another board in the same format.
88         """
89
90         def __init__(self, board, goal_state=None):
91             self.width = len(board[0])
92             self.board = board
93
94             # Default goal state
95             if goal_state is None:
96                 self.goal = [[1, 2, 3],
97                             [4, 5, 6],
98                             [7, 8, 0]]
99             else:
100                self.goal = goal_state
101
102             # Precompute goal positions for Manhattan
103             self.goal_pos = {}
104             for i in range(self.width):
105                 for j in range(self.width):
106                     v = self.goal[i][j]
107                     self.goal_pos[v] = (i, j)
108
109         def solved(self):
110             """
111                 The puzzle is solved if the current board equals the goal board.
112             """
113             return self.board == self.goal
114
115         def actions(self):
116             """
117                 Return a list of 'move', 'action' pairs. 'move' can be called
118                 to return a new puzzle that results in sliding the '0' tile in
119                 the direction of 'action'.
120
121             [1, 2, 3]
122             [4, 5, 6]
123             [7, 8, 0]
124
125             [1, 2, 3]
126             [4, 5, 6]
127             [7, 8, 0]
128
129             [1, 2, 3]
130             [4, 5, 6]
131             [7, 8, 0]
132
133             [1, 2, 3]
134             [4, 5, 6]
135             [7, 8, 0]
136
137             [1, 2, 3]
138             [4, 5, 6]
139             [7, 8, 0]
```

powershell X

```
(.ipy) PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms> & "D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms.jupyter\Scripts\python.exe" "d:/1.SQM/1.UNIVERSIDAD/5. QUINTO SEMESTRE/2.Advanced Algorithms/Homework14_Puzzle/A01571222_A01722925_8puzzle.py"
```

```
None
[1, 2, 3]
[4, 5, 6]
[6, 7, 8]

U
[1, 2, 0]
[4, 5, 3]
[6, 7, 8]

L
[1, 0, 2]
[4, 5, 3]
[6, 7, 8]

D
[1, 5, 2]
[4, 0, 3]
[6, 7, 8]

L
[1, 5, 2]
[0, 4, 3]
[6, 7, 8]

U
[0, 5, 2]
[1, 4, 3]
[6, 7, 8]

R
[5, 0, 2]
[1, 4, 3]
[6, 7, 8]

R
[5, 2, 0]
[1, 4, 3]
[6, 7, 8]

D
[5, 2, 3]
[1, 4, 0]
[6, 7, 8]

D
[5, 2, 3]
[1, 4, 8]
[6, 7, 0]

L
[5, 2, 3]
```

File Edit Selection View Go Run Terminal Help

Q 2.Advanced Algorithms

A01571222_A01722925_8puzzle.py U

```
Homework14_Puzzle > A01571222_A01722925_8puzzle.py > Node > __init__
```

```
101 class Puzzle:
102     def actions(self):
103         """ to return a new puzzle that results in sliding the '0' tile in
104             the direction of 'action'.
105             """
106
107         def create_move(at, to):
108             return lambda: self._move(at, to)
109
110         moves = []
111         for i, j in itertools.product(range(self.width),
112                                       range(self.width)):
113             direcs = {'R': (i, j - 1),
114                       'L': (i, j + 1),
115                       'D': (i - 1, j),
116                       'U': (i + 1, j)}
117
118             for action, (r, c) in direcs.items():
119                 if 0 <= r < self.width and 0 <= c < self.width and self.board[r][c] == 0:
120                     move = create_move((i, j), (r, c)), action
121                     moves.append(move)
122
123         return moves
124
125     def manhattan(self):
126         """ Manhattan distance to the current goal_state.
127         Uses goal_pos computed from self.goal.
128         """
129
130         distance = 0
131         for i in range(self.width):
132             for j in range(self.width):
133                 v = self.board[i][j]
134                 if v == 0:
135                     continue # usually ignore blank
136                 gi, gj = self.goal_pos[v]
137                 distance += abs(gi - i) + abs(gj - j)
138
139         return distance
140
141     def copy(self):
142         """ Return a new puzzle with the same board as 'self'.
143         """
144         board = []
145         for row in self.board:
146             board.append([x for x in row])
147         return Puzzle(board, goal_state=self.goal)
```

(.jupyter) PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms> & "D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms.jupyter\Scripts\python.exe" "d:/1.SQM/1.UNIVERSIDAD/5. QUINTO SEMESTRE/2.Advanced Algorithms/Homework14_Puzzle/A01571222_A01722925_8puzzle.py"

powershell X

```
None
[1, 2, 3]
[4, 5, 0]
[6, 7, 8]

U
[1, 2, 0]
[4, 5, 3]
[6, 7, 8]

L
[1, 0, 2]
[4, 5, 3]
[6, 7, 8]

D
[1, 5, 2]
[4, 0, 3]
[6, 7, 8]

L
[1, 5, 2]
[0, 4, 3]
[6, 7, 8]

U
[0, 5, 2]
[1, 4, 3]
[6, 7, 8]

R
[5, 0, 2]
[1, 4, 3]
[6, 7, 8]

R
[5, 2, 0]
[1, 4, 3]
[6, 7, 8]

D
[5, 2, 3]
[1, 4, 0]
[6, 7, 8]

L
[5, 2, 3]
```

File Edit Selection View Go Run Terminal Help

2.Advanced Algorithms

A01571222_A01722925_8puzzle.py U

Homework14_Puzzle > A01571222_A01722925_8puzzle.py > Node > _init_

```
101 class Puzzle:
102     def __init__(self, board, goal_state=[1, 2, 3], goal_state=[4, 5, 0], goal_state=[6, 7, 8]):
103         self.board = board
104         self.goal_state = goal_state
105         self.actions = {'U': (0, 1), 'D': (0, -1), 'L': (-1, 0), 'R': (1, 0)}
106
107     def __move(self, at, to):
108         """
109             Return a new puzzle where 'at' and 'to' tiles have been swapped.
110             NOTE: all moves should be 'actions' that have been executed
111         """
112         copy = self.copy()
113         i, j = at
114         r, c = to
115         copy.board[i][j], copy.board[r][c] = copy.board[r][c], copy.board[i][j]
116         return copy
117
118     def pprint(self):
119         for row in self.board:
120             print(row)
121         print()
122
123     def __str__(self):
124         return '\n'.join(map(str, self))
125
126     def __iter__(self):
127         for row in self.board:
128             yield from row
129
130
131     board = [[1, 2, 3], [4, 5, 0], [6, 7, 8]]
132     goal_state = [[5, 7, 3], [1, 0, 8], [6, 2, 4]]
133     puzzle = Puzzle(board, goal_state)
134
135     s = Solver(puzzle)
136     tic = time.perf_counter()
137     p = s.solve()
138     toc = time.perf_counter()
139
140     steps = 0
141     for node in p:
142         print(node.action)
143         node.puzzle pprint()
144         steps += 1
145
146     print("Total number of steps: " + str(steps))
147     print("Total amount of time in search: " + str(toc - tic) + " second(s)")
148
149
```

powershell

```
(.jupyter) PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms> & "D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms\jupyter\Scripts\python.exe" "d:/1.SQM/1.UNIVERSIDAD/5. QUINTO SEMESTRE/2.Advanced Algorithms/Homework14_Puzzle/A01571222_A01722925_8puzzle.py"
```

None

```
[1, 2, 3]
[4, 5, 0]
[6, 7, 8]
```

U

```
[1, 2, 0]
[4, 5, 3]
[6, 7, 8]
```

L

```
[1, 0, 2]
[4, 5, 3]
[6, 7, 8]
```

D

```
[1, 5, 2]
[4, 0, 3]
[6, 7, 8]
```

L

```
[1, 5, 2]
[0, 4, 3]
[6, 7, 8]
```

U

```
[0, 5, 2]
[1, 4, 3]
[6, 7, 8]
```

R

```
[5, 0, 2]
[1, 4, 3]
[6, 7, 8]
```

R

```
[5, 2, 0]
[1, 4, 3]
[6, 7, 8]
```

D

```
[5, 2, 3]
[1, 4, 0]
[6, 7, 8]
```

D

```
[5, 2, 3]
[1, 4, 8]
[6, 7, 0]
```

L

```
[5, 2, 3]
```

File Edit Selection View Go Run Terminal Help

2.Advanced Algorithms

A01571222_A01722925_8puzzle.py U

Homework14_Puzzle > A01571222_A01722925_8puzzle.py > Node > _init_

```
101 class Puzzle:
102     def __init__(self, board, goal_state=[1, 2, 3], goal_state=[[5, 2, 3], [1, 4, 0], [6, 7, 8]]):
103         self.board = board
104         self.goal_state = goal_state
105         self.actions = {'U': (0, 1), 'D': (0, -1), 'L': (-1, 0), 'R': (1, 0)}
106         self.states = {tuple(state): None for state in self.all_states()}
107         self.parent = {}
108         self.state = tuple(self.board)
109         self.step = 0
110         self.solver = None
111         self.solver = Solver(self)
112         self.solver.solve()
113         self.pprint()
114
115     def __str__(self):
116         return '\n'.join([' '.join(str(i) for i in row) for row in self.board])
117
118     def __iter__(self):
119         for row in self.board:
120             yield from row
121
122     def pprint(self):
123         for row in self.board:
124             print(row)
125         print()
126
127     def move(self, at, to):
128         """
129             Return a new puzzle where 'at' and 'to' tiles have been swapped.
130             NOTE: all moves should be 'actions' that have been executed
131         """
132         copy = self.copy()
133         i, j = at
134         r, c = to
135         copy.board[i][j], copy.board[r][c] = copy.board[r][c], copy.board[i][j]
136         return copy
137
138     def solve(self):
139         if self.solver is None:
140             self.solver = Solver(self)
141             self.solver.solve()
142         return self.solver.solution
143
144     def copy(self):
145         return Puzzle(self.board, self.goal_state)
146
147     def all_states(self):
148         states = []
149         for state in self.states:
150             states.append([list(state)])
151         return states
152
153     def actions(self, state):
154         actions = []
155         for action, (r, c) in self.actions.items():
156             if state[r][c] == 0:
157                 actions.append(action)
158         return actions
159
160     def result(self, state, action):
161         if action not in self.actions:
162             raise ValueError("Action not found")
163         r, c = self.actions[action]
164         new_state = state.copy()
165         new_state[r][c], new_state[r + r][c + c] = new_state[r + r][c + c], new_state[r][c]
166         return new_state
167
168     def heuristic(self, state):
169         h = 0
170         for i in range(3):
171             for j in range(3):
172                 if state[i][j] != self.goal_state[i][j]:
173                     h += 1
174         return h
175
176     def __eq__(self, other):
177         return self.state == other.state
178
179     def __hash__(self):
180         return hash(tuple(self.state))
181
182     def __lt__(self, other):
183         return self.heuristic(self) < other.heuristic(other)
184
185     def __gt__(self, other):
186         return self.heuristic(self) > other.heuristic(other)
187
188     def __le__(self, other):
189         return self.heuristic(self) <= other.heuristic(other)
190
191     def __ge__(self, other):
192         return self.heuristic(self) >= other.heuristic(other)
193
194     def __ne__(self, other):
195         return self.heuristic(self) != other.heuristic(other)
196
197     def __repr__(self):
198         return f"Puzzle({self.state}, {self.goal_state})"
199
200     def __iter__(self):
201         for row in self.state:
202             yield from row
203
204     def __len__(self):
205         return len(self.state)
206
207     def __bool__(self):
208         return self.state == self.goal_state
209
210     def __nonzero__(self):
211         return self.state == self.goal_state
212
213     def __int__(self):
214         return int(self.state == self.goal_state)
215
216     def __float__(self):
217         return float(self.state == self.goal_state)
218
219     def __complex__(self):
220         return complex(self.state == self.goal_state)
221
222     def __str__(self):
223         return str(self.state)
224
225     def __repr__(self):
226         return repr(self.state)
227
228     def __hash__(self):
229         return hash(self.state)
230
231     def __eq__(self, other):
232         return self.state == other.state
233
234     def __ne__(self, other):
235         return self.state != other.state
236
237     def __lt__(self, other):
238         return self.state < other.state
239
240     def __gt__(self, other):
241         return self.state > other.state
242
243     def __le__(self, other):
244         return self.state <= other.state
245
246     def __ge__(self, other):
247         return self.state >= other.state
248
249     def __int__(self):
250         return int(self.state == self.goal_state)
251
252     def __float__(self):
253         return float(self.state == self.goal_state)
254
255     def __complex__(self):
256         return complex(self.state == self.goal_state)
257
258     def __str__(self):
259         return str(self.state)
260
261     def __repr__(self):
262         return repr(self.state)
263
264     def __hash__(self):
265         return hash(self.state)
266
267     def __eq__(self, other):
268         return self.state == other.state
269
270     def __ne__(self, other):
271         return self.state != other.state
272
273     def __lt__(self, other):
274         return self.state < other.state
275
276     def __gt__(self, other):
277         return self.state > other.state
278
279     def __le__(self, other):
280         return self.state <= other.state
281
282     def __ge__(self, other):
283         return self.state >= other.state
284
285     def __int__(self):
286         return int(self.state == self.goal_state)
287
288     def __float__(self):
289         return float(self.state == self.goal_state)
290
291     def __complex__(self):
292         return complex(self.state == self.goal_state)
293
294     def __str__(self):
295         return str(self.state)
296
297     def __repr__(self):
298         return repr(self.state)
299
300     def __hash__(self):
301         return hash(self.state)
302
303     def __eq__(self, other):
304         return self.state == other.state
305
306     def __ne__(self, other):
307         return self.state != other.state
308
309     def __lt__(self, other):
310         return self.state < other.state
311
312     def __gt__(self, other):
313         return self.state > other.state
314
315     def __le__(self, other):
316         return self.state <= other.state
317
318     def __ge__(self, other):
319         return self.state >= other.state
320
321     def __int__(self):
322         return int(self.state == self.goal_state)
323
324     def __float__(self):
325         return float(self.state == self.goal_state)
326
327     def __complex__(self):
328         return complex(self.state == self.goal_state)
329
330     def __str__(self):
331         return str(self.state)
332
333     def __repr__(self):
334         return repr(self.state)
335
336     def __hash__(self):
337         return hash(self.state)
338
339     def __eq__(self, other):
340         return self.state == other.state
341
342     def __ne__(self, other):
343         return self.state != other.state
344
345     def __lt__(self, other):
346         return self.state < other.state
347
348     def __gt__(self, other):
349         return self.state > other.state
350
351     def __le__(self, other):
352         return self.state <= other.state
353
354     def __ge__(self, other):
355         return self.state >= other.state
356
357     def __int__(self):
358         return int(self.state == self.goal_state)
359
360     def __float__(self):
361         return float(self.state == self.goal_state)
362
363     def __complex__(self):
364         return complex(self.state == self.goal_state)
365
366     def __str__(self):
367         return str(self.state)
368
369     def __repr__(self):
370         return repr(self.state)
371
372     def __hash__(self):
373         return hash(self.state)
374
375     def __eq__(self, other):
376         return self.state == other.state
377
378     def __ne__(self, other):
379         return self.state != other.state
380
381     def __lt__(self, other):
382         return self.state < other.state
383
384     def __gt__(self, other):
385         return self.state > other.state
386
387     def __le__(self, other):
388         return self.state <= other.state
389
390     def __ge__(self, other):
391         return self.state >= other.state
392
393     def __int__(self):
394         return int(self.state == self.goal_state)
395
396     def __float__(self):
397         return float(self.state == self.goal_state)
398
399     def __complex__(self):
400         return complex(self.state == self.goal_state)
401
402     def __str__(self):
403         return str(self.state)
404
405     def __repr__(self):
406         return repr(self.state)
407
408     def __hash__(self):
409         return hash(self.state)
410
411     def __eq__(self, other):
412         return self.state == other.state
413
414     def __ne__(self, other):
415         return self.state != other.state
416
417     def __lt__(self, other):
418         return self.state < other.state
419
420     def __gt__(self, other):
421         return self.state > other.state
422
423     def __le__(self, other):
424         return self.state <= other.state
425
426     def __ge__(self, other):
427         return self.state >= other.state
428
429     def __int__(self):
430         return int(self.state == self.goal_state)
431
432     def __float__(self):
433         return float(self.state == self.goal_state)
434
435     def __complex__(self):
436         return complex(self.state == self.goal_state)
437
438     def __str__(self):
439         return str(self.state)
440
441     def __repr__(self):
442         return repr(self.state)
443
444     def __hash__(self):
445         return hash(self.state)
446
447     def __eq__(self, other):
448         return self.state == other.state
449
450     def __ne__(self, other):
451         return self.state != other.state
452
453     def __lt__(self, other):
454         return self.state < other.state
455
456     def __gt__(self, other):
457         return self.state > other.state
458
459     def __le__(self, other):
460         return self.state <= other.state
461
462     def __ge__(self, other):
463         return self.state >= other.state
464
465     def __int__(self):
466         return int(self.state == self.goal_state)
467
468     def __float__(self):
469         return float(self.state == self.goal_state)
470
471     def __complex__(self):
472         return complex(self.state == self.goal_state)
473
474     def __str__(self):
475         return str(self.state)
476
477     def __repr__(self):
478         return repr(self.state)
479
480     def __hash__(self):
481         return hash(self.state)
482
483     def __eq__(self, other):
484         return self.state == other.state
485
486     def __ne__(self, other):
487         return self.state != other.state
488
489     def __lt__(self, other):
490         return self.state < other.state
491
492     def __gt__(self, other):
493         return self.state > other.state
494
495     def __le__(self, other):
496         return self.state <= other.state
497
498     def __ge__(self, other):
499         return self.state >= other.state
500
501     def __int__(self):
502         return int(self.state == self.goal_state)
503
504     def __float__(self):
505         return float(self.state == self.goal_state)
506
507     def __complex__(self):
508         return complex(self.state == self.goal_state)
509
510     def __str__(self):
511         return str(self.state)
512
513     def __repr__(self):
514         return repr(self.state)
515
516     def __hash__(self):
517         return hash(self.state)
518
519     def __eq__(self, other):
520         return self.state == other.state
521
522     def __ne__(self, other):
523         return self.state != other.state
524
525     def __lt__(self, other):
526         return self.state < other.state
527
528     def __gt__(self, other):
529         return self.state > other.state
530
531     def __le__(self, other):
532         return self.state <= other.state
533
534     def __ge__(self, other):
535         return self.state >= other.state
536
537     def __int__(self):
538         return int(self.state == self.goal_state)
539
540     def __float__(self):
541         return float(self.state == self.goal_state)
542
543     def __complex__(self):
544         return complex(self.state == self.goal_state)
545
546     def __str__(self):
547         return str(self.state)
548
549     def __repr__(self):
550         return repr(self.state)
551
552     def __hash__(self):
553         return hash(self.state)
554
555     def __eq__(self, other):
556         return self.state == other.state
557
558     def __ne__(self, other):
559         return self.state != other.state
560
561     def __lt__(self, other):
562         return self.state < other.state
563
564     def __gt__(self, other):
565         return self.state > other.state
566
567     def __le__(self, other):
568         return self.state <= other.state
569
570     def __ge__(self, other):
571         return self.state >= other.state
572
573     def __int__(self):
574         return int(self.state == self.goal_state)
575
576     def __float__(self):
577         return float(self.state == self.goal_state)
578
579     def __complex__(self):
580         return complex(self.state == self.goal_state)
581
582     def __str__(self):
583         return str(self.state)
584
585     def __repr__(self):
586         return repr(self.state)
587
588     def __hash__(self):
589         return hash(self.state)
590
591     def __eq__(self, other):
592         return self.state == other.state
593
594     def __ne__(self, other):
595         return self.state != other.state
596
597     def __lt__(self, other):
598         return self.state < other.state
599
600     def __gt__(self, other):
601         return self.state > other.state
602
603     def __le__(self, other):
604         return self.state <= other.state
605
606     def __ge__(self, other):
607         return self.state >= other.state
608
609     def __int__(self):
610         return int(self.state == self.goal_state)
611
612     def __float__(self):
613         return float(self.state == self.goal_state)
614
615     def __complex__(self):
616         return complex(self.state == self.goal_state)
617
618     def __str__(self):
619         return str(self.state)
620
621     def __repr__(self):
622         return repr(self.state)
623
624     def __hash__(self):
625         return hash(self.state)
626
627     def __eq__(self, other):
628         return self.state == other.state
629
630     def __ne__(self, other):
631         return self.state != other.state
632
633     def __lt__(self, other):
634         return self.state < other.state
635
636     def __gt__(self, other):
637         return self.state > other.state
638
639     def __le__(self, other):
640         return self.state <= other.state
641
642     def __ge__(self, other):
643         return self.state >= other.state
644
645     def __int__(self):
646         return int(self.state == self.goal_state)
647
648     def __float__(self):
649         return float(self.state == self.goal_state)
650
651     def __complex__(self):
652         return complex(self.state == self.goal_state)
653
654     def __str__(self):
655         return str(self.state)
656
657     def __repr__(self):
658         return repr(self.state)
659
660     def __hash__(self):
661         return hash(self.state)
662
663     def __eq__(self, other):
664         return self.state == other.state
665
666     def __ne__(self, other):
667         return self.state != other.state
668
669     def __lt__(self, other):
670         return self.state < other.state
671
672     def __gt__(self, other):
673         return self.state > other.state
674
675     def __le__(self, other):
676         return self.state <= other.state
677
678     def __ge__(self, other):
679         return self.state >= other.state
680
681     def __int__(self):
682         return int(self.state == self.goal_state)
683
684     def __float__(self):
685         return float(self.state == self.goal_state)
686
687     def __complex__(self):
688         return complex(self.state == self.goal_state)
689
690     def __str__(self):
691         return str(self.state)
692
693     def __repr__(self):
694         return repr(self.state)
695
696     def __hash__(self):
697         return hash(self.state)
698
699     def __eq__(self, other):
700         return self.state == other.state
701
702     def __ne__(self, other):
703         return self.state != other.state
704
705     def __lt__(self, other):
706         return self.state < other.state
707
708     def __gt__(self, other):
709         return self.state > other.state
710
711     def __le__(self, other):
712         return self.state <= other.state
713
714     def __ge__(self, other):
715         return self.state >= other.state
716
717     def __int__(self):
718         return int(self.state == self.goal_state)
719
720     def __float__(self):
721         return float(self.state == self.goal_state)
722
723     def __complex__(self):
724         return complex(self.state == self.goal_state)
725
726     def __str__(self):
727         return str(self.state)
728
729     def __repr__(self):
730         return repr(self.state)
731
732     def __hash__(self):
733         return hash(self.state)
734
735     def __eq__(self, other):
736         return self.state == other.state
737
738     def __ne__(self, other):
739         return self.state != other.state
740
741     def __lt__(self, other):
742         return self.state < other.state
743
744     def __gt__(self, other):
745         return self.state > other.state
746
747     def __le__(self, other):
748         return self.state <= other.state
749
750     def __ge__(self, other):
751         return self.state >= other.state
752
753     def __int__(self):
754         return int(self.state == self.goal_state)
755
756     def __float__(self):
757         return float(self.state == self.goal_state)
758
759     def __complex__(self):
760         return complex(self.state == self.goal_state)
761
762     def __str__(self):
763         return str(self.state)
764
765     def __repr__(self):
766         return repr(self.state)
767
768     def __hash__(self):
769         return hash(self.state)
770
771     def __eq__(self, other):
772         return self.state == other.state
773
774     def __ne__(self, other):
775         return self.state != other.state
776
777     def __lt__(self, other):
778         return self.state < other.state
779
780     def __gt__(self, other):
781         return self.state > other.state
782
783     def __le__(self, other):
784         return self.state <= other.state
785
786     def __ge__(self, other):
787         return self.state >= other.state
788
789     def __int__(self):
790         return int(self.state == self.goal_state)
791
792     def __float__(self):
793         return float(self.state == self.goal_state)
794
795     def __complex__(self):
796         return complex(self.state == self.goal_state)
797
798     def __str__(self):
799         return str(self.state)
800
801     def __repr__(self):
802         return repr(self.state)
803
804     def __hash__(self):
805         return hash(self.state)
806
807     def __eq__(self, other):
808         return self.state == other.state
809
810     def __ne__(self, other):
811         return self.state != other.state
812
813     def __lt__(self, other):
814         return self.state < other.state
815
816     def __gt__(self, other):
817         return self.state > other.state
818
819     def __le__(self, other):
820         return self.state <= other.state
821
822     def __ge__(self, other):
823         return self.state >= other.state
824
825     def __int__(self):
826         return int(self.state == self.goal_state)
827
828     def __float__(self):
829         return float(self.state == self.goal_state)
830
831     def __complex__(self):
832         return complex(self.state == self.goal_state)
833
834     def __str__(self):
835         return str(self.state)
836
837     def __repr__(self):
838         return repr(self.state)
839
840     def __hash__(self):
841         return hash(self.state)
842
843     def __eq__(self, other):
844         return self.state == other.state
845
846     def __ne__(self, other):
847         return self.state != other.state
848
849     def __lt__(self, other):
850         return self.state < other.state
851
852     def __gt__(self, other):
853         return self.state > other.state
854
855     def __le__(self, other):
856         return self.state <= other.state
857
858     def __ge__(self, other):
859         return self.state >= other.state
860
861     def __int__(self):
862         return int(self.state == self.goal_state)
863
864     def __float__(self):
865         return float(self.state == self.goal_state)
866
867     def __complex__(self):
868         return complex(self.state == self.goal_state)
869
870     def __str__(self):
871         return str(self.state)
872
873     def __repr__(self):
874         return repr(self.state)
875
876     def __hash__(self):
877         return hash(self.state)
878
879     def __eq__(self, other):
880         return self.state == other.state
881
882     def __ne__(self, other):
883         return self.state != other.state
884
885     def __lt__(self, other):
886         return self.state < other.state
887
888     def __gt__(self, other):
889         return self.state > other.state
890
891     def __le__(self, other):
892         return self.state <= other.state
893
894     def __ge__(self, other):
895         return self.state >= other.state
896
897     def __int__(self):
898         return int(self.state == self.goal_state)
899
900     def __float__(self):
901         return float(self.state == self.goal_state)
902
903     def __complex__(self):
904         return complex(self.state == self.goal_state)
905
906     def __str__(self):
907         return str(self.state)
908
909     def __repr__(self):
910         return repr(self.state)
911
912     def __hash__(self):
913         return hash(self.state)
914
915     def __eq__(self, other):
916         return self.state == other.state
917
918     def __ne__(self, other):
919         return self.state != other.state
920
921     def __lt__(self, other):
922         return self.state < other.state
923
924     def __gt__(self, other):
925         return self.state > other.state
926
927     def __le__(self, other):
928         return self.state <= other.state
929
930     def __ge__(self, other):
931         return self.state >= other.state
932
933     def __int__(self):
934         return int(self.state == self.goal_state)
935
936     def __float__(self):
937         return float(self.state == self.goal_state)
938
939     def __complex__(self):
940         return complex(self.state == self.goal_state)
941
942     def __str__(self):
943         return str(self.state)
944
945     def __repr__(self):
946         return repr(self.state)
947
948     def __hash__(self):
949         return hash(self.state)
950
951     def __eq__(self, other):
952         return self.state == other.state
953
954     def __ne__(self, other):
955         return self.state != other.state
956
957     def __lt__(self, other):
958         return self.state < other.state
959
960     def __gt__(self, other):
961         return self.state > other.state
962
963     def __le__(self, other):
964         return self.state <= other.state
965
966     def __ge__(self, other):
967         return self.state >= other.state
968
969     def __int__(self):
970         return int(self.state == self.goal_state)
971
972     def __float__(self):
973         return float(self.state == self.goal_state)
974
975     def __complex__(self):
976         return complex(self.state == self.goal_state)
977
978     def __str__(self):
979         return str(self.state)
980
981     def __repr__(self):
982         return repr(self.state)
983
984     def __hash__(self):
985         return hash(self.state)
986
987     def __eq__(self, other):
988         return self.state == other.state
989
990     def __ne__(self, other):
991         return self.state != other.state
992
993     def __lt__(self, other):
994         return self.state < other.state
995
996     def __gt__(self, other):
997         return self.state > other.state
998
999     def __le__(self, other):
1000        return self.state <= other.state
1001
1002        def __ge__(self, other):
1003            return self.state >= other.state
1004
1005        def __int__(self):
1006            return int(self.state == self.goal_state)
1007
1008        def __float__(self):
1009            return float(self.state == self.goal_state)
1010
1011        def __complex__(self):
1012            return complex(self.state == self.goal_state)
1013
1014        def __str__(self):
1015            return str(self.state)
1016
1017        def __repr__(self):
1018            return repr(self.state)
1019
1020        def __hash__(self):
1021            return hash(self.state)
1022
1023        def __eq__(self, other):
1024            return self.state == other.state
1025
1026        def __ne__(self, other):
1027            return self.state != other.state
1028
1029        def __lt__(self, other):
1030            return self.state < other.state
1031
1032        def __gt__(self, other):
1033            return self.state > other.state
1034
1035        def __le__(self, other):
1036            return self.state <= other.state
1037
1038        def __ge__(self, other):
1039            return self.state >= other.state
1040
1041        def __int__(self):
1042            return int(self.state == self.goal_state)
1043
1044        def __float__(self):
1045            return float(self.state == self.goal_state)
1046
1047        def __complex__(self):
1048            return complex(self.state == self.goal_state)
1049
1050        def __str__(self):
1051            return str(self.state)
1052
1053        def __repr__(self):
1054            return repr(self.state)
1055
1056        def __hash__(self):
1057            return hash(self.state)
1058
1059        def __eq__(self, other):
1060            return self.state == other.state
1061
1062        def __ne__(self, other):
1063            return self.state != other.state
1064
1065        def __lt__(self, other):
1066            return self.state < other.state
1067
1068        def __gt__(self, other):
1069            return self.state > other.state
1070
1071        def __le__(self, other):
1072            return self.state <= other.state
1073
1074        def __ge__(self, other):
1075            return self.state >= other.state
1076
1077        def __int__(self):
1078            return int(self.state == self.goal_state)
1079
1080        def __float__(self):
1081            return float(self.state == self.goal_state)
1082
1083        def __complex__(self):
1084            return complex(self.state == self.goal_state)
1085
1086        def __str__(self):
1087            return str(self.state)
1088
1089        def __repr__(self):
1090            return repr(self.state)
1091
1092        def __hash__(self):
1093            return hash(self.state)
1094
1095        def __eq__(self, other):
1096            return self.state == other.state
1097
1098        def __ne__(self, other):
1099            return self.state != other.state
1100
1101        def __lt__(self, other):
1102            return self.state < other.state
1103
1104        def __gt__(self, other):
1105            return self.state > other.state
1106
1107        def __le__(self, other):
1108            return self.state <= other.state
1109
1110        def __ge__(self, other):
1111            return self.state >= other.state
1112
1113        def __int__(self):
1114            return int(self.state == self.goal_state)
1115
1116        def __float__(self):
1117            return float(self.state == self.goal_state)
1118
1119        def __complex__(self):
1120            return complex(self.state == self.goal_state)
1121
1122        def __str__(self):
1123            return str(self.state)
1124
1125        def __repr__(self):
1126            return repr(self.state)
1127
1128        def __hash__(self):
1129            return hash(self.state)
1130
1131        def __eq__(self, other):
1132            return self.state == other.state
1133
1134        def __ne__(self, other):
1135            return self.state != other.state
1136
1137        def __lt__(self, other):
1138            return self.state < other.state
1139
1140        def __gt__(self, other):
1141            return self.state > other.state
1142
1143        def __le__(self, other):
1144            return self.state <= other.state
1145
1146        def __ge__(self, other):
1147            return self.state >= other.state
1148
1149        def __int__(self):
1150            return int(self.state == self.goal_state)
1151
1152        def __float__(self):
1153            return float(self.state == self.goal_state)
1154
1155        def __complex__(self):
1156            return complex(self.state == self.goal_state)
1157
1158        def __str__(self):
1159            return str(self.state)
1160
1161        def __repr__(self):
1162            return repr(self.state)
1163
1164        def __hash__(self):
1165            return hash(self.state)
1166
1167        def __eq__(self, other):
1168            return self.state == other.state
1169
1170        def __ne__(self, other):
1171            return self.state != other.state
1172
1173        def __lt__(self, other):
1174            return self.state < other.state
1175
1176        def __gt__(self, other):
1177            return self.state > other.state
1178
1179        def __le__(self, other):
1180            return self.state <= other.state
1181
1182        def __ge__(self, other):
1183            return self.state >= other.state
1184
1185        def __int__(self):
1186            return int(self.state == self.goal_state)
1187
1188        def __float__(self):
1189            return float(self.state == self.goal_state)
1190
1191        def __complex__(self):
1192            return complex(self.state == self.goal_state)
1193
1194        def __str__(self):
1195            return str(self.state)
1196
1197        def __repr__(self):
1198            return repr(self.state)
1199
1200        def __hash__(self):
1201            return hash(self.state)
1202
1203        def __eq__(self, other):
1204            return self.state == other.state
1205
1206        def __ne__(self, other):
1207            return self.state != other.state
1208
1209        def __lt__(self, other):
1210            return self.state < other.state
1211
1212        def __gt__(self, other):
1213            return self.state > other.state
1214
1215        def __le__(self, other):
1216            return self.state <= other.state
1217
1218        def __ge__(self, other):
1219            return self.state >= other.state
1220
1221        def __int__(self):
1222            return int(self.state == self.goal_state)
1223
1224        def __float__(self):
1225            return float(self.state == self.goal_state)
1226
1227        def __complex__(self):
1228            return complex(self.state == self.goal_state)
1229
1230        def __str__(self):
1231            return str(self.state)
1232
1233        def __repr__(self):
1234            return repr(self.state)
1235
1236        def __hash__(self):
1237            return hash(self.state)
1238
1239        def __eq__(self, other):
1240            return self.state == other.state
1241
1242        def __ne__(self, other):
1243            return self.state != other.state
1244
1245        def __lt__(self, other):
1246            return self.state < other.state
1247
1248        def __gt__(self, other):
1249            return self.state > other.state
1250
1251        def __le__(self, other):
1252            return self.state <= other.state
1253
1254        def __ge__(self, other):
1255            return self.state >= other.state
1256
1257        def __int__(self):
1258            return int(self.state == self.goal_state)
1259
1260        def __float__(self):
1261            return float(self.state == self.goal_state)
1262
1263        def __complex__(self):
1264            return complex(self.state == self.goal_state)
1265
1266        def __str__(self):
1267            return str(self.state)
1268
1269        def __repr__(self):
1270            return repr(self.state)
1271
1272        def __hash__(self):
1273            return hash(self.state)
1274
1275        def __eq__(self, other):
1276            return self.state == other.state
1277
1278        def __ne__(self, other):
1279            return self.state != other.state
1280
1281        def __lt__(self, other):
1282            return self.state < other.state
1283
1284        def __gt__(self, other):
1285            return self.state > other.state
1286
1287        def __le__(self, other):
1288            return self.state <= other.state
1289
1290        def __ge__(self, other):
1291            return self.state >= other.state
1292
1293        def __int__(self):
1294            return int(self.state == self.goal_state)
1295
1296        def __float__(self):
1297            return float(self.state == self.goal_state)
1298
1299        def __complex__(self):
1300            return complex(self.state == self.goal_state)
1301
1302        def __str__(self):
1303            return str(self.state)
1304
1305        def __repr__(self):
1306            return repr(self.state)
1307
1308        def __hash__(self):
1309            return hash(self.state)
1310
1311        def __eq__(self, other):
1312            return self.state == other.state
1313
1314        def __ne__(self, other):
1315            return self.state != other.state
1316
1317        def __lt__(self, other):
1318            return self.state < other.state
1319
1320        def __gt__(self, other):
1321            return self.state > other.state
1322
1323        def __le__(self, other):
1324            return self.state <= other.state
1325
1326        def __ge__(self, other):
1327            return self.state >= other.state
1328
1329        def __int__(self):
1330            return int(self.state == self.goal_state)
1331
1332        def __float__(self):
1333            return float(self.state == self.goal_state)
1334
1335        def __complex__(self):
1336            return complex(self.state == self.goal_state)
1337
1338        def __str__(self):
1339            return str(self.state)
1340
1341        def __repr__(self):
1342            return repr(self.state)
1343
1344        def __hash__(self):
1345            return hash(self.state)
1346
1347        def __eq__(self, other):
1348            return self.state == other.state
1349
1350        def __ne__(self, other):
1351            return self.state != other.state
1352
1353        def __lt__(self, other):
1354            return self.state < other.state
1355
1356        def __gt__(self, other):
1357            return self.state > other.state
1358
1359        def __le__(self, other):
1360            return self.state <= other.state
1361
1362        def __ge__(self, other):
1363            return self.state >= other.state
1364
1365        def __int__(self):
1366            return int(self.state == self.goal_state)
1367
1368        def __float__(self):
1369            return float(self.state == self.goal_state)
1370
1371        def __complex__(self):
1372            return complex(self.state == self.goal_state)
1373
1374        def __str__(self):
1375            return str(self.state)
1376
1377        def __repr__(self):
1378            return repr(self.state)
1379
1380        def __hash__(self):
1381            return hash(self.state)
1382
1383        def __eq__(self, other):
1384            return self.state == other.state
1385
1386        def __ne__(self, other):
1387            return self.state != other.state
1388
1389        def __lt__(self, other):
1390            return self.state < other.state
1391
1392        def __gt__(self, other):
1393            return self.state > other.state
1394
1395        def __le__(self, other):
1396            return self.state <= other.state
1397
1398        def __ge__(self, other):
1399            return self.state >= other.state
1400
1401        def __int__(self):
1402            return int(self.state == self.goal_state)
1403
1404        def __float__(self):
1405           
```

File Edit Selection View Go Run Terminal Help

2. Advanced Algorithms

A01571222_A01722925_8puzzle.py U

Homework14_Puzzle > A01571222_A01722925_8puzzle.py > Node > _init_

```
101 class Puzzle:
102     def __init__(self, board, goal_state):
103         self.board = board
104         self.goal_state = goal_state
105         self.actions = {'L': (0, 1), 'R': (0, -1), 'U': (-1, 0), 'D': (1, 0)}
106
107     def __move(self, at, to):
108         """Return a new puzzle where 'at' and 'to' tiles have been swapped.
109         NOTE: all moves should be 'actions' that have been executed
110         """
111         copy = self.copy()
112         i, j = at
113         r, c = to
114         copy.board[i][j], copy.board[r][c] = copy.board[r][c], copy.board[i][j]
115         return copy
116
117     def pprint(self):
118         for row in self.board:
119             print(row)
120         print()
121
122     def __str__(self):
123         return '\n'.join(map(str, self))
124
125     def __iter__(self):
126         for row in self.board:
127             yield from row
128
129
130     board = [[1, 2, 3], [4, 5, 0], [6, 7, 8]]
131     goal_state = [[5, 7, 3], [1, 0, 8], [6, 2, 4]]
132     puzzle = Puzzle(board, goal_state)
133
134     s = Solver(puzzle)
135     tic = time.perf_counter()
136     p = s.solve()
137     toc = time.perf_counter()
138
139     steps = 0
140     for node in p:
141         print(node.action)
142         node.puzzle pprint()
143         steps += 1
144
145     print("Total number of steps: " + str(steps))
146     print("Total amount of time in search: " + str(toc - tic) + " second(s)")
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
```

powershell X

```
L
[5, 2, 3]
[1, 4, 8]
[6, 0, 7]

U
"""
[5, 2, 3]
[1, 0, 8]
[6, 4, 7]

U
[5, 0, 3]
[1, 2, 8]
[6, 4, 7]

R
[5, 3, 8]
[1, 2, 8]
[6, 4, 7]

D
[5, 3, 8]
[1, 2, 0]
[6, 4, 7]

D
[5, 3, 8]
[1, 2, 7]
[6, 4, 0]

L
[5, 3, 8]
[1, 2, 7]
[6, 0, 4]

U
[5, 3, 8]
[1, 0, 7]
[6, 2, 4]

R
[5, 3, 8]
[1, 7, 0]
[6, 2, 4]

U
[5, 3, 0]
[1, 7, 8]
[6, 2, 4]

L
[5, 0, 3]
[1, 7, 8]
[6, 2, 4]
```

File Edit Selection View Go Run Terminal Help

Q 2.Advanced Algorithms

A01571222_A01722925_8puzzle.py U

Homework14_Puzzle > A01571222_A01722925_8puzzle.py > Node > _init_

```
101 class Puzzle:
102     def __init__(self, board, goal_state=[1, 2, 3, 4, 5, 6, 7, 8]):
103         self.board = board
104         self.goal_state = goal_state
105         self.actions = {'U': (0, 1), 'D': (0, -1), 'L': (-1, 0), 'R': (1, 0)}
106         self.states = {tuple(self.board): None}
107         self.parent = {}
108         self.step = 0
109
110     def __str__(self):
111         return '\n'.join([' '.join(str(i) for i in row) for row in self.board])
112
113     def __iter__(self):
114         for row in self.board:
115             yield row
116
117     def move(self, at, to):
118         copy = self.copy()
119         i, j = at
120         r, c = to
121         copy.board[i][j], copy.board[r][c] = copy.board[r][c], copy.board[i][j]
122         return copy
123
124     def pprint(self):
125         for row in self.board:
126             print(row)
127         print()
128
129     def __repr__(self):
130         return str(self)
131
132     def solve(self):
133         if tuple(self.board) in self.states:
134             return None
135         self.states[tuple(self.board)] = self
136         if self.board == self.goal_state:
137             return self
138         for direction in self.actions:
139             new_board = self.move((0, 1), (1, 0))
140             if tuple(new_board) not in self.states:
141                 new_puzzle = self.__class__(new_board, self.goal_state)
142                 new_puzzle.parent = self
143                 new_puzzle.step = self.step + 1
144                 self.parent[new_puzzle] = direction
145                 self.states[tuple(new_board)] = new_puzzle
146
147         return self.solve()
148
149     def copy(self):
150         return [row[:] for row in self.board]
151
152     def __eq__(self, other):
153         return self.board == other.board
154
155     def __hash__(self):
156         return hash(tuple(self.board))
157
158     def __lt__(self, other):
159         return self.step < other.step
160
161     def __gt__(self, other):
162         return self.step > other.step
163
164     def __le__(self, other):
165         return self.step <= other.step
166
167     def __ge__(self, other):
168         return self.step >= other.step
169
170     def __ne__(self, other):
171         return self.step != other.step
172
173     def __lt__(self, other):
174         return self.step < other.step
175
176     def __gt__(self, other):
177         return self.step > other.step
178
179     def __le__(self, other):
180         return self.step <= other.step
181
182     def __ge__(self, other):
183         return self.step >= other.step
184
185     def __ne__(self, other):
186         return self.step != other.step
187
188     def __lt__(self, other):
189         return self.step < other.step
190
191     def __gt__(self, other):
192         return self.step > other.step
193
194     def __le__(self, other):
195         return self.step <= other.step
196
197     def __ge__(self, other):
198         return self.step >= other.step
199
200     def __ne__(self, other):
201         return self.step != other.step
202
203     def __lt__(self, other):
204         return self.step < other.step
205
206     def __gt__(self, other):
207         return self.step > other.step
208
209     def __le__(self, other):
210         return self.step <= other.step
211
212     def __ge__(self, other):
213         return self.step >= other.step
214
215     def __ne__(self, other):
216         return self.step != other.step
217
218     def __lt__(self, other):
219         return self.step < other.step
220
221     def __gt__(self, other):
222         return self.step > other.step
223
224     def __le__(self, other):
225         return self.step <= other.step
226
227     def __ge__(self, other):
228         return self.step >= other.step
229
230     def __ne__(self, other):
231         return self.step != other.step
232
233     def __lt__(self, other):
234         return self.step < other.step
235
236     def __gt__(self, other):
237         return self.step > other.step
238
239     def __le__(self, other):
240         return self.step <= other.step
241
242     def __ge__(self, other):
243         return self.step >= other.step
244
245     def __ne__(self, other):
246         return self.step != other.step
247
248     def __lt__(self, other):
249         return self.step < other.step
250
251     def __gt__(self, other):
252         return self.step > other.step
253
254     def __le__(self, other):
255         return self.step <= other.step
256
257     def __ge__(self, other):
258         return self.step >= other.step
259
260     def __ne__(self, other):
261         return self.step != other.step
262
263     def __lt__(self, other):
264         return self.step < other.step
265
266     def __gt__(self, other):
267         return self.step > other.step
268
269     def __le__(self, other):
270         return self.step <= other.step
271
272     def __ge__(self, other):
273         return self.step >= other.step
274
275     def __ne__(self, other):
276         return self.step != other.step
277
278     def __lt__(self, other):
279         return self.step < other.step
280
281     def __gt__(self, other):
282         return self.step > other.step
283
284     def __le__(self, other):
285         return self.step <= other.step
286
287     def __ge__(self, other):
288         return self.step >= other.step
289
290     def __ne__(self, other):
291         return self.step != other.step
292
293     def __lt__(self, other):
294         return self.step < other.step
295
296     def __gt__(self, other):
297         return self.step > other.step
298
299     def __le__(self, other):
300         return self.step <= other.step
301
302     def __ge__(self, other):
303         return self.step >= other.step
304
305     def __ne__(self, other):
306         return self.step != other.step
307
308     def __lt__(self, other):
309         return self.step < other.step
310
311     def __gt__(self, other):
312         return self.step > other.step
313
314     def __le__(self, other):
315         return self.step <= other.step
316
317     def __ge__(self, other):
318         return self.step >= other.step
319
320     def __ne__(self, other):
321         return self.step != other.step
322
323     def __lt__(self, other):
324         return self.step < other.step
325
326     def __gt__(self, other):
327         return self.step > other.step
328
329     def __le__(self, other):
330         return self.step <= other.step
331
332     def __ge__(self, other):
333         return self.step >= other.step
334
335     def __ne__(self, other):
336         return self.step != other.step
337
338     def __lt__(self, other):
339         return self.step < other.step
340
341     def __gt__(self, other):
342         return self.step > other.step
343
344     def __le__(self, other):
345         return self.step <= other.step
346
347     def __ge__(self, other):
348         return self.step >= other.step
349
350     def __ne__(self, other):
351         return self.step != other.step
352
353     def __lt__(self, other):
354         return self.step < other.step
355
356     def __gt__(self, other):
357         return self.step > other.step
358
359     def __le__(self, other):
360         return self.step <= other.step
361
362     def __ge__(self, other):
363         return self.step >= other.step
364
365     def __ne__(self, other):
366         return self.step != other.step
367
368     def __lt__(self, other):
369         return self.step < other.step
370
371     def __gt__(self, other):
372         return self.step > other.step
373
374     def __le__(self, other):
375         return self.step <= other.step
376
377     def __ge__(self, other):
378         return self.step >= other.step
379
380     def __ne__(self, other):
381         return self.step != other.step
382
383     def __lt__(self, other):
384         return self.step < other.step
385
386     def __gt__(self, other):
387         return self.step > other.step
388
389     def __le__(self, other):
390         return self.step <= other.step
391
392     def __ge__(self, other):
393         return self.step >= other.step
394
395     def __ne__(self, other):
396         return self.step != other.step
397
398     def __lt__(self, other):
399         return self.step < other.step
400
401     def __gt__(self, other):
402         return self.step > other.step
403
404     def __le__(self, other):
405         return self.step <= other.step
406
407     def __ge__(self, other):
408         return self.step >= other.step
409
410     def __ne__(self, other):
411         return self.step != other.step
412
413     def __lt__(self, other):
414         return self.step < other.step
415
416     def __gt__(self, other):
417         return self.step > other.step
418
419     def __le__(self, other):
420         return self.step <= other.step
421
422     def __ge__(self, other):
423         return self.step >= other.step
424
425     def __ne__(self, other):
426         return self.step != other.step
427
428     def __lt__(self, other):
429         return self.step < other.step
430
431     def __gt__(self, other):
432         return self.step > other.step
433
434     def __le__(self, other):
435         return self.step <= other.step
436
437     def __ge__(self, other):
438         return self.step >= other.step
439
440     def __ne__(self, other):
441         return self.step != other.step
442
443     def __lt__(self, other):
444         return self.step < other.step
445
446     def __gt__(self, other):
447         return self.step > other.step
448
449     def __le__(self, other):
450         return self.step <= other.step
451
452     def __ge__(self, other):
453         return self.step >= other.step
454
455     def __ne__(self, other):
456         return self.step != other.step
457
458     def __lt__(self, other):
459         return self.step < other.step
460
461     def __gt__(self, other):
462         return self.step > other.step
463
464     def __le__(self, other):
465         return self.step <= other.step
466
467     def __ge__(self, other):
468         return self.step >= other.step
469
470     def __ne__(self, other):
471         return self.step != other.step
472
473     def __lt__(self, other):
474         return self.step < other.step
475
476     def __gt__(self, other):
477         return self.step > other.step
478
479     def __le__(self, other):
480         return self.step <= other.step
481
482     def __ge__(self, other):
483         return self.step >= other.step
484
485     def __ne__(self, other):
486         return self.step != other.step
487
488     def __lt__(self, other):
489         return self.step < other.step
490
491     def __gt__(self, other):
492         return self.step > other.step
493
494     def __le__(self, other):
495         return self.step <= other.step
496
497     def __ge__(self, other):
498         return self.step >= other.step
499
500     def __ne__(self, other):
501         return self.step != other.step
502
503     def __lt__(self, other):
504         return self.step < other.step
505
506     def __gt__(self, other):
507         return self.step > other.step
508
509     def __le__(self, other):
510         return self.step <= other.step
511
512     def __ge__(self, other):
513         return self.step >= other.step
514
515     def __ne__(self, other):
516         return self.step != other.step
517
518     def __lt__(self, other):
519         return self.step < other.step
520
521     def __gt__(self, other):
522         return self.step > other.step
523
524     def __le__(self, other):
525         return self.step <= other.step
526
527     def __ge__(self, other):
528         return self.step >= other.step
529
530     def __ne__(self, other):
531         return self.step != other.step
532
533     def __lt__(self, other):
534         return self.step < other.step
535
536     def __gt__(self, other):
537         return self.step > other.step
538
539     def __le__(self, other):
540         return self.step <= other.step
541
542     def __ge__(self, other):
543         return self.step >= other.step
544
545     def __ne__(self, other):
546         return self.step != other.step
547
548     def __lt__(self, other):
549         return self.step < other.step
550
551     def __gt__(self, other):
552         return self.step > other.step
553
554     def __le__(self, other):
555         return self.step <= other.step
556
557     def __ge__(self, other):
558         return self.step >= other.step
559
560     def __ne__(self, other):
561         return self.step != other.step
562
563     def __lt__(self, other):
564         return self.step < other.step
565
566     def __gt__(self, other):
567         return self.step > other.step
568
569     def __le__(self, other):
570         return self.step <= other.step
571
572     def __ge__(self, other):
573         return self.step >= other.step
574
575     def __ne__(self, other):
576         return self.step != other.step
577
578     def __lt__(self, other):
579         return self.step < other.step
580
581     def __gt__(self, other):
582         return self.step > other.step
583
584     def __le__(self, other):
585         return self.step <= other.step
586
587     def __ge__(self, other):
588         return self.step >= other.step
589
589 Total number of steps: 22
590 Total amount of time in search: 0.2641739000027883 second(s)

powershell x



```
[6, 4, 7]
U
[5, 0, 3]
[1, 2, 8]
[6, 4, 7]
R
[5, 3, 0]
[1, 2, 8]
[6, 4, 7]
D
[5, 3, 8]
[1, 2, 0]
[6, 4, 0]
D
[5, 3, 8]
[1, 2, 7]
[6, 0, 4]
U
[5, 3, 8]
[1, 0, 7]
[6, 2, 4]
R
[5, 3, 8]
[1, 7, 0]
[6, 2, 4]
U
[5, 3, 0]
[1, 7, 8]
[6, 2, 4]
L
[5, 0, 3]
[1, 7, 8]
[6, 2, 4]
D
[5, 7, 3]
[1, 0, 8]
[6, 2, 4]
```


```

https://colab.research.google.com/drive/14YoHYCqee8-XPWc_uEUSPSjNXZ6M_Oa5?usp=sharing