**8 Puzzle problem or fixing the grid/matrix using Iterative Deepening Search (IDS)**

[[Kapil Lanjewar](https://medium.com/@kapil.lanjewar.100?source=post_page-----11c81af86681--------------------------------)](https://medium.com/@kapil.lanjewar.100?source=post_page-----11c81af86681--------------------------------)

[Kapil Lanjewar](https://medium.com/@kapil.lanjewar.100?source=post_page-----11c81af86681--------------------------------)

3 min read

·

Aug 27

3

Given input matrix:

| 3, 2, 1 |  
| 4, 5, 6 |  
| 8, 7, 0 |  
  
===================================================  
  
Array representation -> [3, 2, 1, 4, 5, 6, 8, 7, 0]

Expected output matrix to be fixed:

| 8, 3, 1 |  
| 5, 4, 6 |  
| 0, 7, 2 |  
  
===================================================  
  
Array representation -> [8, 3, 1, 5, 4, 6, 0, 7, 2]

**Note: Here “0” is used for blank or empty space on the board**

*Python source code*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*main.py*

import resource  
import sys  
  
from iddfs import IDDFS  
from board import Board  
  
def main():  
 t = [8, 3, 1, 5, 4, 6, 0, 7, 2]  
 c = [3, 2, 1, 4, 5, 6, 8, 7, 0]  
   
 p = Board(c)  
 p.set\_target(t)  
   
 steps = 0  
   
 s = IDDFS(p, steps)  
 s.solve()  
   
 steps = s.number\_of\_steps()  
 print("Number of moves")  
 print(steps)  
  
 print("Exiting program")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

*board.py*

import numpy as np  
  
class Board:  
 parent = None  
 state = None  
 operator = None  
 depth = 0  
 zero = None  
 cost = 0  
  
 def \_\_init\_\_(self, state, parent=None, operator=None, depth=0):  
 self.parent = parent  
 self.state = np.array(state)  
 self.operator = operator  
 self.depth = depth  
 self.zero = self.find\_0()  
 self.cost = self.depth + self.manhattan()  
  
 def \_\_lt\_\_(self, other):  
 if self.cost != other.cost:  
 return self.cost < other.cost  
 else:  
 op\_pr = {'Up': 0, 'Down': 1, 'Left': 2, 'Right': 3}  
 return op\_pr[self.operator] < op\_pr[other.operator]  
  
 def \_\_str\_\_(self):  
 return str(self.state[:3]) + '\n' \  
 + str(self.state[3:6]) + '\n' \  
 + str(self.state[6:]) + ' ' + str(self.depth) + str(self.operator) + '\n'  
  
 def set\_target(self, t):  
 global target  
 target = t  
  
 def goal\_test(self):  
 print("--")  
 print(target)  
 print(self.state)  
 print("xx")  
 if np.array\_equal(self.state, target):  
 return True  
 else:  
 return False  
  
 def find\_0(self):  
 for i in range(9):  
 if self.state[i] == 0:  
 return i  
  
 def manhattan(self):  
 state = self.index(self.state)  
 goal = self.index(np.arange(9))  
 return sum((abs(state // 3 - goal // 3) + abs(state % 3 - goal % 3))[1:])  
  
 @staticmethod  
 def index(state):  
 index = np.array(range(9))  
 for x, y in enumerate(state):  
 index[y] = x  
 return index  
  
 def swap(self, i, j):  
 new\_state = np.array(self.state)  
 new\_state[i], new\_state[j] = new\_state[j], new\_state[i]  
 return new\_state  
  
 def up(self):  
 if self.zero > 2:  
 return Board(self.swap(self.zero, self.zero - 3), self, 'Up', self.depth + 1)  
 else:  
 return None  
  
 def down(self):  
 if self.zero < 6:  
 return Board(self.swap(self.zero, self.zero + 3), self, 'Down', self.depth + 1)  
 else:  
 return None  
  
 def left(self):  
 if self.zero % 3 != 0:  
 return Board(self.swap(self.zero, self.zero - 1), self, 'Left', self.depth + 1)  
 else:  
 return None  
  
 def right(self):  
 if (self.zero + 1) % 3 != 0:  
 return Board(self.swap(self.zero, self.zero + 1), self, 'Right', self.depth + 1)  
 else:  
 return None  
  
 def neighbors(self):  
 neighbors = [self.up(), self.down(), self.left(), self.right()]  
 return list(filter(None, neighbors))  
  
 \_\_repr\_\_ = \_\_str\_\_

*solver.py*

from abc import ABC, abstractmethod  
  
  
class Solver(ABC):  
 solution = None  
 frontier = None  
 nodes\_expanded = 0  
 max\_depth = 0  
 explored\_nodes = set()  
 initial\_state = None  
  
 def \_\_init\_\_(self, initial\_state):  
 self.initial\_state = initial\_state  
  
 def ancestral\_chain(self):  
 current = self.solution  
 chain = [current]  
 while current.parent is not None:  
 chain.append(current.parent)  
 current = current.parent  
 return chain  
  
 @property  
 def path(self):  
 path = [node.operator for node in self.ancestral\_chain()[-2::-1]]  
 return path  
  
 @abstractmethod  
 def solve(self):  
 pass  
  
 def set\_solution(self, board):  
 self.solution = board  
 self.nodes\_expanded = len(self.explored\_nodes) - len(self.frontier) - 1

*iddfs.py*

import itertools  
  
from solver import Solver  
  
  
class IDDFS(Solver):  
 def \_\_init\_\_(self, initial\_state, steps):  
 super(IDDFS, self).\_\_init\_\_(initial\_state)  
 self.frontier = []  
 self.steps = steps  
  
 def dls(self, limit):  
 self.frontier.append(self.initial\_state)  
 while self.frontier:  
 board = self.frontier.pop()  
 self.explored\_nodes.add(tuple(board.state))  
 if board.goal\_test():  
 self.set\_solution(board)  
 return self.solution  
 if board.depth < limit:  
 for neighbor in board.neighbors()[::-1]:  
 if tuple(neighbor.state) not in self.explored\_nodes:  
 self.frontier.append(neighbor)  
 self.explored\_nodes.add(tuple(neighbor.state))  
 self.max\_depth = max(self.max\_depth, neighbor.depth)  
 return None  
  
 def solve(self):  
 for i in itertools.count():  
 self.frontier = []  
 self.explored\_nodes = set()  
 self.max\_depth = 0  
 self.frontier.append(self.initial\_state)  
 sol = self.dls(i)  
 self.steps = i  
 if sol is not None:  
 break  
 return  
   
 def number\_of\_steps(self):  
 return self.steps

*Output of the program*

*—  
[8, 3, 1, 5, 4, 6, 0, 7, 2]  
[8 3 1 5 4 6 0 7 2]  
xx*

*Number of moves  
16  
Exiting program*

[Python](https://medium.com/tag/python?source=post_page-----11c81af86681---------------python-----------------)

[Algorithms](https://medium.com/tag/algorithms?source=post_page-----11c81af86681---------------algorithms-----------------)

[Artificial Intelligence](https://medium.com/tag/artificial-intelligence?source=post_page-----11c81af86681---------------artificial_intelligence-----------------)