**8 Puzzle problem or fixing the grid/matrix using Uniform Cost Search (UCS)**

[[Kapil Lanjewar](https://medium.com/@kapil.lanjewar.100?source=post_page-----f7de8e96c140--------------------------------)](https://medium.com/@kapil.lanjewar.100?source=post_page-----f7de8e96c140--------------------------------)

[Kapil Lanjewar](https://medium.com/@kapil.lanjewar.100?source=post_page-----f7de8e96c140--------------------------------)

4 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:

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

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

Python source code

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

main.py

# goal\_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]  
goal\_state = [2, 5, 1, 3, 6, 0, 4, 8, 7]  
# goal\_state = [3, 2, 1, 4, 5, 6, 8, 0, 7]  
  
counter = 0  
  
  
def display\_board(state):  
 print("-------------")  
 print("| %i | %i | %i |" % (state[0], state[3], state[6]))  
 print("-------------")  
 print("| %i | %i | %i |" % (state[1], state[4], state[7]))  
 print("-------------")  
 print("| %i | %i | %i |" % (state[2], state[5], state[8]))  
 print("-------------")  
  
  
def move\_up(state):  
 """Moves the blank tile up on the board. Returns a new state as a list."""  
 # Perform an object copy  
 new\_state = state[:]  
 index = new\_state.index(0)  
 # Sanity check  
 if index not in [0, 3, 6]:  
 # Swap the values.  
 temp = new\_state[index - 1]  
 new\_state[index - 1] = new\_state[index]  
 new\_state[index] = temp  
 return new\_state  
 else:  
 # Can't move, return None (Pythons NULL)  
 return None  
  
  
def move\_down(state):  
 """Moves the blank tile down on the board. Returns a new state as a list."""  
 # Perform object copy  
 new\_state = state[:]  
 index = new\_state.index(0)  
 # Sanity check  
 if index not in [2, 5, 8]:  
 # Swap the values.  
 temp = new\_state[index + 1]  
 new\_state[index + 1] = new\_state[index]  
 new\_state[index] = temp  
 return new\_state  
 else:  
 # Can't move, return None.  
 return None  
  
  
def move\_left(state):  
 """Moves the blank tile left on the board. Returns a new state as a list."""  
 new\_state = state[:]  
 index = new\_state.index(0)  
 # Sanity check  
 if index not in [0, 1, 2]:  
 # Swap the values.  
 temp = new\_state[index - 3]  
 new\_state[index - 3] = new\_state[index]  
 new\_state[index] = temp  
 return new\_state  
 else:  
 # Can't move it, return None  
 return None  
  
  
def move\_right(state):  
 """Moves the blank tile right on the board. Returns a new state as a list."""  
 # Performs an object copy. Python passes by reference.  
 new\_state = state[:]  
 index = new\_state.index(0)  
 # Sanity check  
 if index not in [6, 7, 8]:  
 # Swap the values.  
 temp = new\_state[index + 3]  
 new\_state[index + 3] = new\_state[index]  
 new\_state[index] = temp  
 return new\_state  
 else:  
 # Can't move, return None  
 return None  
  
  
def create\_node(state, parent, operator, depth, cost):  
 return Node(state, parent, operator, depth, cost)  
  
  
def expand\_node(node):  
 """Returns a list of expanded nodes"""  
 expanded\_nodes = [create\_node(move\_up(node.state), node, "u", node.depth + 1, 0),  
 create\_node(move\_down(node.state), node, "d", node.depth + 1, 0),  
 create\_node(move\_left(node.state), node, "l", node.depth + 1, 0),  
 create\_node(move\_right(node.state), node, "r", node.depth + 1, 0)]  
 # Filter the list and remove the nodes that are impossible (move function returned None)  
 expanded\_nodes = [node for node in expanded\_nodes if node.state is not None] # list comprehension!  
 return expanded\_nodes  
  
  
def bfs(start, goal):  
 """Performs a breadth first search from the start state to the goal"""  
 # A list (can act as a queue) for the nodes.  
 goal = goal  
 start\_node = create\_node(start, None, None, 0, 0)  
 fringe = [start\_node]  
 current = fringe.pop(0)  
 path = []  
 while current.state != goal:  
 fringe.extend(expand\_node(current))  
 current = fringe.pop(0)  
 while current.parent is not None:  
 path.insert(0, current.operator)  
 current = current.parent  
 return path  
 pass  
  
  
def dfs(start, goal, depth=10):  
 start\_node = create\_node(start, None, None, 0, 0)  
 fringe\_stack = [start\_node]  
 current = fringe\_stack.pop()  
 path = []  
 while current.state != goal:  
 temp = expand\_node(current)  
 for item in temp:  
 fringe\_stack.extend(expand\_node(item))  
 current = fringe\_stack.pop()  
 if current.depth > 10:  
 return None  
 while current.parent is not None:  
 path.insert(0, current.operator)  
 current = current.parent  
 return path  
  
  
def uniform\_cost(start, goal, increase):  
 start\_node = create\_node(start, None, None, 0, 0)  
 fringe = []  
 path = []  
 fringe.append(start\_node)  
 current = fringe.pop(0)  
 while current.state != goal:  
 increase = increase + 1  
 print(goal)  
 print(current.state)  
 print(increase)  
 temp = expand\_node(current)  
 for item in temp:  
 item.depth += current.depth  
 fringe.append(item)  
 fringe.sort(key=lambda x: x.depth)  
 current = fringe.pop(0)  
 if increase > 200000:  
 break  
 while current.parent is not None:  
 path.insert(0, current.operator)  
 current = current.parent  
 return path  
  
  
def greedy(start, goal):  
 start\_node = create\_node(start, None, None, 0, 0)  
 fringe = []  
 path = []  
 fringe.append(start\_node)  
 current = fringe.pop(0)  
 while current.state != goal:  
 fringe.extend(expand\_node(current))  
 for item in fringe:  
 h(item, goal)  
 fringe.sort(key=lambda x: x.heuristic)  
 current = fringe.pop(0)  
 while current.parent is not None:  
 path.insert(0, current.operator)  
 current = current.parent  
 return path  
  
  
def a\_star(start, goal):  
 start\_node = create\_node(start, None, None, 0, 0)  
 fringe = []  
 path = []  
 fringe.append(start\_node)  
 current = fringe.pop(0)  
 while current.state != goal:  
 fringe.extend(expand\_node(current))  
 for item in fringe:  
 h(item, goal)  
 item.heuristic += item.depth  
 fringe.sort(key=lambda x: x.heuristic)  
 current = fringe.pop(0)  
 while current.parent is not None:  
 path.insert(0, current.operator)  
 current = current.parent  
 return path  
  
  
def h(state, goal):  
 dmatch = 0  
 for i in range(0, 9):  
 if state.state[i] != goal[i]:  
 dmatch += 1  
 state.heuristic = dmatch  
  
  
# Node data structure  
class Node:  
 def \_\_init\_\_(self, state, parent, operator, depth, cost):  
 # Contains the state of the node  
 self.state = state  
 # Contains the node that generated this node  
 self.parent = parent  
 # Contains the operation that generated this node from the parent  
 self.operator = operator  
 # Contains the depth of this node (parent.depth +1)  
 self.depth = depth  
 # Contains the path cost of this node from depth 0. Not used for depth/breadth first.  
 self.cost = cost  
  
 self.heuristic = None  
  
  
def readfile(filename):  
 f = open(filename)  
 data = f.read()  
 # Get rid of the newlines  
 data = data.strip("\n")  
 # Break the string into a list using a space as a seperator.  
 data = data.split(" ")  
 state = []  
 for element in data:  
 state.append(int(element))  
 print('state: ', state)  
 return state  
  
  
# Main method  
def main():  
 starting\_state = [3, 2, 1, 4, 5, 6, 8, 7, 0]  
  
 result = uniform\_cost(starting\_state, goal\_state, counter)  
   
 if result is None:  
 print("No solution found")  
 elif result == [None]:  
 print("Start node was the goal!")  
 else:  
 print(result)  
 print(len(result), " moves")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

*Output of the program:*

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

[Python](https://medium.com/tag/python?source=post_page-----f7de8e96c140---------------python-----------------)

[Algorithms](https://medium.com/tag/algorithms?source=post_page-----f7de8e96c140---------------algorithms-----------------)

[Artificial Intelligence](https://medium.com/tag/artificial-intelligence?source=post_page-----f7de8e96c140---------------artificial_intelligence-----------------)