Artificial Intelligence Practical 4

18BCE259: Shrey Viradiya

AIM: Implement the 8 puzzle problem using A* Algorithm

8 puzzle problem is about reaching the final state from any intermediary state. Final State is defined as below:

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 |
```

Misplaced Tiles

This heuristic function returns integer representing the number of tiles no in the correct possition

Manhattan Distance

This heuristic function returns integer giving sum of manhatten distance of the each tile with its correct position

```
import numpy as np

class Node:
    def __init__(self, data, parent, act, g, h_ = 0):
        self.data = data
        self.parent = parent
        self.act = act
        self.g = g
        self.h_ = h_

    def __str__(self):
        return self.data.__repr__()

initial_node = Node(np.array([[1,2,3],[4,0,6],[7,5,8]]), None, None,
0, 0)

def whereEmpty(node):
    return np.argwhere(node.data == 0)[0]

whereEmpty(initial_node)

array([1, 1])

print(initial_node)
```

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

def misplaced_tiles(data):
    return 9 - np.sum(data == np.array([[1,2,3],[4,5,6],[7,8,0]]))

def manhattan_distance(data):
    place = {
        0: (2,2),
        1: (0,0),
        2: (0,1),
        3: (0,2),
        4: (1,0),
        5: (1,1),
        6: (1,2),
        7: (2,0),
        8: (2,1)
    }
    md = 0
    for (x,y),d in np.ndenumerate(data):
        md += abs(place[d][0] - x) + abs(place[d][1] - y)
    return md

def canMoveLeft(node):
    return whereEmpty(node)[1] != 0

def moveLeft(node, h_function):
    pointer = whereEmpty(node)
    new_Node = Node(np.copy(node.data), node, 'left', node.g + 1)

    temp = new_Node.data[pointer[0], pointer[1] - 1]
    new_Node.data[pointer[0], pointer[1] - 1] =
new_Node.data[pointer[0], pointer[1]]
    new_Node.data[pointer[0], pointer[1]] = temp

    new_Node.h_ = h_function(new_Node.data)

    return new_Node

print(canMoveLeft(initial_node))
print(moveLeft(initial_node, misplaced_tiles))

True
array([[1, 2, 3],
       [0, 4, 6],
       [7, 5, 8]])

def canMoveRight(node):
    return whereEmpty(node)[1] != 2
```

```
def moveRight(node, h_function):
    pointer = whereEmpty(node)
    new_Node = Node(np.copy(node.data), node, 'right', node.g + 1)

    temp = new_Node.data[pointer[0], pointer[1] + 1]
    new_Node.data[pointer[0], pointer[1] + 1] =
new_Node.data[pointer[0], pointer[1]]
    new_Node.data[pointer[0], pointer[1]] = temp

    new_Node.h_ = h_function(new_Node.data)

    return new_Node

print(canMoveRight(initial_node))
print(moveRight(initial_node, misplaced_tiles))
```

```
True
array([[1, 2, 3],
       [4, 6, 0],
       [7, 5, 8]])
```

```
def canMoveUp(node):
    return whereEmpty(node)[0] != 0

def moveUp(node, h_function):
    pointer = whereEmpty(node)

    new_Node = Node(np.copy(node.data), node, 'up', node.g + 1)


    temp = new_Node.data[pointer[0] - 1, pointer[1]]
    new_Node.data[pointer[0] - 1, pointer[1]] =
new_Node.data[pointer[0], pointer[1]]
    new_Node.data[pointer[0], pointer[1]] = temp

    new_Node.h_ = h_function(new_Node.data)

    return new_Node

print(canMoveUp(initial_node))
print(moveUp(initial_node, misplaced_tiles))
```

```
True
array([[1, 0, 3],
       [4, 2, 6],
       [7, 5, 8]])
```

```
def canMoveDown(node):
    return whereEmpty(node)[0] != 2
```

```
def moveDown(node, h_function):
    pointer = whereEmpty(node)
    new_Node = Node(np.copy(node.data), node, 'down', node.g + 1)

    temp = new_Node.data[pointer[0] + 1, pointer[1]]
    new_Node.data[pointer[0] + 1, pointer[1]] =
new_Node.data[pointer[0], pointer[1]]
    new_Node.data[pointer[0], pointer[1]] = temp

    new_Node.h_ = h_function(new_Node.data)

    return new_Node

print(canMoveDown(initial_node))
print(moveDown(initial_node, misplaced_tiles))
```

```
True
array([[1, 2, 3],
       [4, 5, 6],
       [7, 0, 8]])
```

```
def CheckIfFinal(node):
    return np.all(node.data == np.array([[1,2,3],[4,5,6],[7,8,0]]))

CheckIfFinal(initial_node)
```

```
False
```

```
# Python3 program to check if a given
# instance of 8 puzzle is solvable or not

# A utility function to count
# inversions in given array 'arr[]'

def getInvCount(arr):
    inv_count = 0
    empty_value = -1
    for i in range(0, 9):
        for j in range(i + 1, 9):
            if arr[j] != empty_value and arr[i] != empty_value and
arr[i] > arr[j]:
                inv_count += 1
    return inv_count


# This function returns true
# if given 8 puzzle is solvable.
def isSolvable(puzzle) :

    # Count inversions in given 8 puzzle
    inv_count = getInvCount([j for sub in puzzle for j in sub])
```

```
    # return true if inversion count is even.
    return (inv_count % 2 == 0)

    # Driver code
puzzle = [[4, 1, 0], [7, 2, 3], [5, 8, 6]]
if(isSolvable(puzzle)) :
    print("Solvable")
else :
    print("Not Solvable")
```

```
Solvable
```

```
import random

def generatePuzzle(dep):
    node = Node(np.array([[1,2,3],[4,5,6],[7,8,0]]), None, None, 0)

    while (node.g != dep):
        step = random.randint(1,4)

        if step == 1 and canMoveLeft(node) and node.act != "right":
            node = moveLeft(node, misplaced_tiles)
        elif step == 2 and canMoveRight(node) and node.act != "left":
            node = moveRight(node, misplaced_tiles)
        elif step == 3 and canMoveUp(node) and node.act != "down":
            node = moveUp(node, misplaced_tiles)
        elif step == 4 and canMoveDown(node) and node.act != "up":
            node = moveDown(node, misplaced_tiles)
        else:
            continue
        print(node)

generatePuzzle(50)
```

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

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

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

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

```
def A_star(initial_node, h_function):

    if(not isSolvable(initial_node.data)):
        print("Puzzle Not Solvable")
        return

    Closed = []
    Opened = [initial_node]

    min_node = None

    while True:

        if(len(Opened) == 0):
```

```
        return False

    min_node = Opened[0]
    for n in Opened:
        if ((n.g+n.h_) < (min_node.g+min_node.h_)):
            min_node = n

    Opened.remove(min_node)
    Closed.append(min_node)

    if(CheckIfFinal(min_node)):
        print("Final State Obtained")
        break

    if canMoveLeft(min_node) and min_node.act != "right":
        node_l = moveLeft(min_node, h_function)
        if(CheckIfFinal(node_l)):
            print("Final State Obtained")
            min_node = node_l
            break
        Opened.append(node_l)
    if canMoveRight(min_node) and min_node.act != "left":
        node_r = moveRight(min_node, h_function)
        if(CheckIfFinal(node_r)):
            print("Final State Obtained")
            min_node = node_r
            break
        Opened.append(node_r)
    if canMoveUp(min_node) and min_node.act != "down":
        node_u = moveUp(min_node, h_function)
        if(CheckIfFinal(node_u)):
            print("Final State Obtained")
            min_node = node_u
            break
        Opened.append(node_u)
    if canMoveDown(min_node) and min_node.act != "up":
        node_d = moveDown(min_node, h_function)
        if(CheckIfFinal(node_d)):
            print("Final State Obtained")
            min_node = node_d
            break
        Opened.append(node_d)

moves = []
while (min_node != None):
    moves.insert(0, min_node.act)
    min_node = min_node.parent

print(f"Solution: {len(moves) - 1} moves")
print("==========================")
```

```
    print("Initial Node: ")

    node = initial_node
    for move in moves:
        print(move)
        if move == 'left':
            node = moveLeft(node, misplaced_tiles)
        elif move == 'right':
            node = moveRight(node, misplaced_tiles)
        elif move == 'up':
            node = moveUp(node, misplaced_tiles)
        elif move == 'down':
            node = moveDown(node, misplaced_tiles)
        print(node)

    return True

A_star(initial_node, misplaced_tiles)
```

```
Final State Obtained
Solution: 2 moves
============================
Initial Node:
None
array([[1, 2, 3],
       [4, 0, 6],
       [7, 5, 8]])
down
array([[1, 2, 3],
       [4, 5, 6],
       [7, 0, 8]])
right
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])

True
```

```
%%time
initial_node = Node(np.array([[0,1,3],[4,2,6],[7,5,8]]), None, None,
0)
A_star(initial_node, misplaced_tiles)
```

```
Final State Obtained
Solution: 4 moves
============================
Initial Node:
None
array([[0, 1, 3],
       [4, 2, 6],
       [7, 5, 8]])
```

```
right
array([[1, 0, 3],
       [4, 2, 6],
       [7, 5, 8]])
down
array([[1, 2, 3],
       [4, 0, 6],
       [7, 5, 8]])
down
array([[1, 2, 3],
       [4, 5, 6],
       [7, 0, 8]])
right
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])
CPU times: user 4.7 ms, sys: 149 µs, total: 4.85 ms
Wall time: 5.48 ms

True
```

```
%%time
initial_node = Node(np.array([[4, 3, 6], [2, 0, 1], [7, 5, 8]]), None,
None, 0)
A_star(initial_node, misplaced_tiles)
```

```
Final State Obtained
Solution: 10 moves
============================
Initial Node:
None
array([[4, 3, 6],
       [2, 0, 1],
       [7, 5, 8]])
right
array([[4, 3, 6],
       [2, 1, 0],
       [7, 5, 8]])
up
array([[4, 3, 0],
       [2, 1, 6],
       [7, 5, 8]])
left
array([[4, 0, 3],
       [2, 1, 6],
       [7, 5, 8]])
down
array([[4, 1, 3],
       [2, 0, 6],
       [7, 5, 8]])
left
```

```
array([[4, 1, 3],
       [0, 2, 6],
       [7, 5, 8]])
up
array([[0, 1, 3],
       [4, 2, 6],
       [7, 5, 8]])
right
array([[1, 0, 3],
       [4, 2, 6],
       [7, 5, 8]])
down
array([[1, 2, 3],
       [4, 0, 6],
       [7, 5, 8]])
down
array([[1, 2, 3],
       [4, 5, 6],
       [7, 0, 8]])
right
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])
CPU times: user 6.82 ms, sys: 7.84 ms, total: 14.7 ms
Wall time: 15.6 ms

True
```

```
%%time
initial_node = Node(np.array([[0,1,2],[4,5,3],[7,8,6]]), None, None,
0)
A_star(initial_node, misplaced_tiles)
```

```
Final State Obtained
Solution: 4 moves
============================
Initial Node:
None
array([[0, 1, 2],
       [4, 5, 3],
       [7, 8, 6]])
right
array([[1, 0, 2],
       [4, 5, 3],
       [7, 8, 6]])
right
array([[1, 2, 0],
       [4, 5, 3],
       [7, 8, 6]])
down
array([[1, 2, 3],
```

```
       [4, 5, 0],
       [7, 8, 6]])
down
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])
CPU times: user 2.29 ms, sys: 3.64 ms, total: 5.93 ms
Wall time: 5.57 ms

True

%%time
initial_node = Node(np.array([[1, 2, 3], [4, 0, 5], [7, 8, 6]]), None,
None, 0)
A_star(initial_node, misplaced_tiles)

Final State Obtained
Solution: 2 moves
============================
Initial Node:
None
array([[1, 2, 3],
       [4, 0, 5],
       [7, 8, 6]])
right
array([[1, 2, 3],
       [4, 5, 0],
       [7, 8, 6]])
down
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])
CPU times: user 3.28 ms, sys: 0 ns, total: 3.28 ms
Wall time: 3.43 ms

True

%%time
initial_node = Node(np.array([[1, 8, 2], [4, 6, 3],[0, 7, 5]]), None,
None, 0)
A_star(initial_node, misplaced_tiles)

Final State Obtained
Solution: 10 moves
============================
Initial Node:
None
array([[1, 8, 2],
       [4, 6, 3],
       [0, 7, 5]])
right
```

```
array([[1, 8, 2],
       [4, 6, 3],
       [7, 0, 5]])
up
array([[1, 8, 2],
       [4, 0, 3],
       [7, 6, 5]])
up
array([[1, 0, 2],
       [4, 8, 3],
       [7, 6, 5]])
right
array([[1, 2, 0],
       [4, 8, 3],
       [7, 6, 5]])
down
array([[1, 2, 3],
       [4, 8, 0],
       [7, 6, 5]])
down
array([[1, 2, 3],
       [4, 8, 5],
       [7, 6, 0]])
left
array([[1, 2, 3],
       [4, 8, 5],
       [7, 0, 6]])
up
array([[1, 2, 3],
       [4, 0, 5],
       [7, 8, 6]])
right
array([[1, 2, 3],
       [4, 5, 0],
       [7, 8, 6]])
down
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])
CPU times: user 13.2 ms, sys: 2.76 ms, total: 16 ms
Wall time: 18.6 ms

True
```

```
%%time
initial_node = Node(np.array([[4, 1, 8],[7, 2, 6],[0, 3, 5]]), None,
None, 0)
A_star(initial_node, manhattan_distance)
```

```
Final State Obtained
Solution: 18 moves
```

```
============================
Initial Node:
None
array([[4, 1, 8],
       [7, 2, 6],
       [0, 3, 5]])
up
array([[4, 1, 8],
       [0, 2, 6],
       [7, 3, 5]])
up
array([[0, 1, 8],
       [4, 2, 6],
       [7, 3, 5]])
right
array([[1, 0, 8],
       [4, 2, 6],
       [7, 3, 5]])
down
array([[1, 2, 8],
       [4, 0, 6],
       [7, 3, 5]])
right
array([[1, 2, 8],
       [4, 6, 0],
       [7, 3, 5]])
up
array([[1, 2, 0],
       [4, 6, 8],
       [7, 3, 5]])
left
array([[1, 0, 2],
       [4, 6, 8],
       [7, 3, 5]])
down
array([[1, 6, 2],
       [4, 0, 8],
       [7, 3, 5]])
down
array([[1, 6, 2],
       [4, 3, 8],
       [7, 0, 5]])
right
array([[1, 6, 2],
       [4, 3, 8],
       [7, 5, 0]])
up
array([[1, 6, 2],
       [4, 3, 0],
       [7, 5, 8]])
```

```
left
array([[1, 6, 2],
       [4, 0, 3],
       [7, 5, 8]])
up
array([[1, 0, 2],
       [4, 6, 3],
       [7, 5, 8]])
right
array([[1, 2, 0],
       [4, 6, 3],
       [7, 5, 8]])
down
array([[1, 2, 3],
       [4, 6, 0],
       [7, 5, 8]])
left
array([[1, 2, 3],
       [4, 0, 6],
       [7, 5, 8]])
down
array([[1, 2, 3],
       [4, 5, 6],
       [7, 0, 8]])
right
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])
CPU times: user 49.7 ms, sys: 186 µs, total: 49.9 ms
Wall time: 51.8 ms

True
```

```
%%time
initial_node = Node(np.array([[3, 5, 0],[8, 7, 2],[6, 4, 1]]), None,
None, 0)
A_star(initial_node, misplaced_tiles)
```

```
Final State Obtained
Solution: 24 moves
==========================
Initial Node:
None
array([[3, 5, 0],
       [8, 7, 2],
       [6, 4, 1]])
down
array([[3, 5, 2],
       [8, 7, 0],
       [6, 4, 1]])
down
```

```
array([[3, 5, 2],
       [8, 7, 1],
       [6, 4, 0]])
left
array([[3, 5, 2],
       [8, 7, 1],
       [6, 0, 4]])
left
array([[3, 5, 2],
       [8, 7, 1],
       [0, 6, 4]])
up
array([[3, 5, 2],
       [0, 7, 1],
       [8, 6, 4]])
right
array([[3, 5, 2],
       [7, 0, 1],
       [8, 6, 4]])
right
array([[3, 5, 2],
       [7, 1, 0],
       [8, 6, 4]])
down
array([[3, 5, 2],
       [7, 1, 4],
       [8, 6, 0]])
left
array([[3, 5, 2],
       [7, 1, 4],
       [8, 0, 6]])
left
array([[3, 5, 2],
       [7, 1, 4],
       [0, 8, 6]])
up
array([[3, 5, 2],
       [0, 1, 4],
       [7, 8, 6]])
right
array([[3, 5, 2],
       [1, 0, 4],
       [7, 8, 6]])
right
array([[3, 5, 2],
       [1, 4, 0],
       [7, 8, 6]])
up
array([[3, 5, 0],
       [1, 4, 2],
```

```
       [7, 8, 6]])
left
array([[3, 0, 5],
       [1, 4, 2],
       [7, 8, 6]])
left
array([[0, 3, 5],
       [1, 4, 2],
       [7, 8, 6]])
down
array([[1, 3, 5],
       [0, 4, 2],
       [7, 8, 6]])
right
array([[1, 3, 5],
       [4, 0, 2],
       [7, 8, 6]])
right
array([[1, 3, 5],
       [4, 2, 0],
       [7, 8, 6]])
up
array([[1, 3, 0],
       [4, 2, 5],
       [7, 8, 6]])
left
array([[1, 0, 3],
       [4, 2, 5],
       [7, 8, 6]])
down
array([[1, 2, 3],
       [4, 0, 5],
       [7, 8, 6]])
right
array([[1, 2, 3],
       [4, 5, 0],
       [7, 8, 6]])
down
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])
CPU times: user 4min 52s, sys: 1.44 s, total: 4min 54s
Wall time: 4min 58s

True
```

```
%%time
initial_node = Node(np.array([[3, 5, 0],[8, 7, 2],[6, 4, 1]]), None,
None, 0)
A_star(initial_node, manhattan_distance)
```

```
Final State Obtained
Solution: 24 moves
==========================
Initial Node:
None
array([[3, 5, 0],
       [8, 7, 2],
       [6, 4, 1]])
down
array([[3, 5, 2],
       [8, 7, 0],
       [6, 4, 1]])
down
array([[3, 5, 2],
       [8, 7, 1],
       [6, 4, 0]])
left
array([[3, 5, 2],
       [8, 7, 1],
       [6, 0, 4]])
left
array([[3, 5, 2],
       [8, 7, 1],
       [0, 6, 4]])
up
array([[3, 5, 2],
       [0, 7, 1],
       [8, 6, 4]])
right
array([[3, 5, 2],
       [7, 0, 1],
       [8, 6, 4]])
right
array([[3, 5, 2],
       [7, 1, 0],
       [8, 6, 4]])
down
array([[3, 5, 2],
       [7, 1, 4],
       [8, 6, 0]])
left
array([[3, 5, 2],
       [7, 1, 4],
       [8, 0, 6]])
left
array([[3, 5, 2],
       [7, 1, 4],
       [0, 8, 6]])
up
array([[3, 5, 2],
```

```
       [0, 1, 4],
       [7, 8, 6]])
right
array([[3, 5, 2],
       [1, 0, 4],
       [7, 8, 6]])
right
array([[3, 5, 2],
       [1, 4, 0],
       [7, 8, 6]])
up
array([[3, 5, 0],
       [1, 4, 2],
       [7, 8, 6]])
left
array([[3, 0, 5],
       [1, 4, 2],
       [7, 8, 6]])
left
array([[0, 3, 5],
       [1, 4, 2],
       [7, 8, 6]])
down
array([[1, 3, 5],
       [0, 4, 2],
       [7, 8, 6]])
right
array([[1, 3, 5],
       [4, 0, 2],
       [7, 8, 6]])
right
array([[1, 3, 5],
       [4, 2, 0],
       [7, 8, 6]])
up
array([[1, 3, 0],
       [4, 2, 5],
       [7, 8, 6]])
left
array([[1, 0, 3],
       [4, 2, 5],
       [7, 8, 6]])
down
array([[1, 2, 3],
       [4, 0, 5],
       [7, 8, 6]])
right
array([[1, 2, 3],
       [4, 5, 0],
       [7, 8, 6]])
```

```
down
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])
CPU times: user 623 ms, sys: 3.97 ms, total: 627 ms
Wall time: 639 ms

True
```