# Artificial Intelligence Practical 2

## 18BCE259: Shrey Viradiya

### AIM: Implement the 8 puzzle problem using BFS

**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 |
```

In [1]:

```
import numpy as np
```

In [2]:

```
class Node:
    def __init__(self, data, parent, act):
        self.data = data
        self.parent = parent
        self.act = act

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

In [3]:

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

In [4]:

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

whereEmpty(initial_node)
```

Out[4]:

```
array([1, 1], dtype=int64)
```

In [5]:

```
print(initial_node)
```

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

In [6]:

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

def moveLeft(node):
    pointer = whereEmpty(node)
    new_Node = Node(np.copy(node.data), node, 'left')
```

```
    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

    return new_Node

print(canMoveLeft(initial_node))
print(moveLeft(initial_node))
```

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

In [7]:

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

def moveRight(node):
    pointer = whereEmpty(node)
    new_Node = Node(np.copy(node.data), node, 'right')

    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

    return new_Node

print(canMoveRight(initial_node))
print(moveRight(initial_node))
```

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

In [8]:

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

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

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


    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

    return new_Node

print(canMoveUp(initial_node))
print(moveUp(initial_node))
```

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

In [9]:

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

def moveDown(node):
    pointer = whereEmpty(node)
    new_Node = Node(np.copy(node.data), node, 'down')

    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

    return new_Node

print(canMoveDown(initial_node))
print(moveDown(initial_node))
```

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

In [10]:

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

CheckIfFinal(initial_node)
```

Out[10]:

```
False
```

In [11]:

```
# 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
    for i in range(0, 2) :
        for j in range(i + 1, 3) :

            # Value 0 is used for empty space
            if (arr[j][i] > 0 and arr[j][i] > arr[i][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
    invCount = getInvCount(puzzle)

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

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

```
Solvable
```

In [12]:

```
def BFS(initial_node):

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

    passed_node = []
    Queue = [initial_node]
```

```
    node = None

    while True:
        node = Queue.pop(0)

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

        if canMoveLeft(node) and node.act != "right":
            Queue.append(moveLeft(node))
        if canMoveRight(node) and node.act != "left":
            Queue.append(moveRight(node))
        if canMoveUp(node) and node.act != "down":
            Queue.append(moveUp(node))
        if canMoveDown(node) and node.act != "up":
            Queue.append(moveDown(node))

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

    print("Solution: ")
    print("==========================")
    print("Initial Node: ")

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

In [13]:

```
BFS(initial_node)
```

```
Final State Obtained
Solution:
==========================
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]])
```

In [14]:

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

```
Final State Obtained
Solution:
==========================
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]])
```

In [15]:

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

```
Puzzle Not Solvable
```

In [16]:

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

```
Puzzle Not Solvable
```

In [17]:

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

In [ ]:

```

```