```
import time

class Node:

    def __init__ (self, puzzle):
        self.children = []
        self.parent = None
        self.puzzle = puzzle
        self.zero = 0
        self.g = 0
        self.f = self.get_f_value()
        self.x = 0

    def print_puzzle(self):
        print()
        m = 0
        for i in range(3):
            for j in range(3):
                print(self.puzzle[m], end=" ")
                m += 1
            print()

    def create_child (self, puzzle):
        child = Node(puzzle)
        self.children.append(child)
        child.parent = self
        child.g = self.g + 1

    def get_f_value(self):
        h = 0
        for i in range(len(self.puzzle)):
            if self.puzzle[i] != i:
                h += 1
        return h + self.g

    def move_right(self):
        if (self.x + 1) % 3 != 0:
            puzzle_child = self.puzzle[:]
            puzzle_child[self.x], puzzle_child[self.x +
                  1] = puzzle_child[self.x+1], puzzle_child[self.x]
            self.create_child(puzzle_child)

    def move_left(self):
        if self.x % 3 != 0:
            puzzle_child = self.puzzle[:]
            puzzle_child[(self.x)], puzzle_child[(self.x) -
                   1] = puzzle_child[(self.x) - 1], puzzle_child[(self.x)]
        self.create_child(puzzle_child)

def move_up(self):
    if self.x > 2:
        puzzle_child = self.puzzle[:]
        puzzle_child[(self.x)], puzzle_child[(self.x) -
           3] = puzzle_child[(self.x) - 3], puzzle_child[(self.x)]
        self.create_child(puzzle_child)

def move_down(self):
    if self.x < 6:
        puzzle_child = self.puzzle[:]
        puzzle_child[(self.x)], puzzle_child[(self.x) +
           3] = puzzle_child[(self.x) + 3], puzzle_child[(self.x)]
        self.create_child(puzzle_child)

def goaltest(self):
    isGoal = True
    for i in range(len(self.puzzle)):
        if i != self.puzzle[i]:
            isGoal = False
             return isGoal
    return isGoal

def expand_node(self):
    for i in range(len(self.puzzle)):
        if self.puzzle[i] == 0:
             self.x = i
    self.move_right()
    self.move_down()
    self.move_left()
    self.move_up()

def is_unsolvable(self):
    print(self.puzzle)
    count = 0

    for i in range(8):
        for j in range(i, 9):
            if self.puzzle[i] > self.puzzle[j] and self.puzzle[j] != 0:
                count += 1
    if count % 2 == 1:
        return True
    else:
        return False
```

```
class Search:

    def __init__(self):
        pass

    def a_star_search(self,root):
        open_list = []
        visited = set()
        open_list.append(root)
        visited.add(tuple(root.puzzle))

        while(True):
            current_Node = open_list.pop(0)

            if current_Node.goaltest():
                pathtosolution = Search.path_trace(current_Node)
                print(len(visited))
                return pathtosolution

            current_Node.expand_node()

            for current_child in current_Node.children:
                if tuple(current_child.puzzle) not in visited:
                    open_list.append(current_child)
                    visited.add(tuple(current_child.puzzle))

            open_list.sort(key=lambda x: x.f)

    def path_trace(n):
        current = n
        path = []
        path.append(current)

        while current.parent != None:
            current = current.parent
            path.append(current)
        return path

if __name__ == "__main__":

 puzzle = [8,6,7,2,5,4,3,0,1]
 root = Node(puzzle)

 if root.is_unsolvable():
    print("Puzzle has no solution")
else:
    s = Search()
    print("Finding solution..")
    start = time.time()
    solution = s.a_star_search(root)
    end = time.time()
    solution.reverse()

    for i in range(len(solution)):
        solution[i].print_puzzle()
    print("Number of steps taken:", len(solution)-1)
    print("Elapsed time:", end-start)
```

Solution:

```
[8, 6, 7, 2, 5, 4, 3, 0, 1]
Finding solution..
1358

8 6 7
2 5 4
3 0 1

8 6 7
2 0 4
3 5 1

8 6 7
2 4 0
3 5 1

8 6 0
2 4 7
3 5 1

8 0 6
2 4 7
3 5 1

0 8 6
2 4 7
3 5 1

2 8 6
0 4 7
3 5 1

2 8 6
3 4 7
0 5 1

2 8 6
3 4 7
```

```
5 0 1

2 8 6
3 4 7
5 1 0

2 8 6
3 4 0
5 1 7

2 8 0
3 4 6
5 1 7

2 0 8
3 4 6
5 1 7

2 4 8
3 0 6
5 1 7

2 4 8
3 1 6
5 0 7

2 4 8
3 1 6
5 7 0

2 4 8
3 1 0
5 7 6

2 4 0
3 1 8
5 7 6

2 0 4
3 1 8
5 7 6

2 1 4
3 0 8
5 7 6

2 1 4
3 7 8
5 0 6

2 1 4
3 7 8
5 6 0

2 1 4
3 7 0
5 6 8
```

```
2 1 0
3 7 4
5 6 8

2 0 1
3 7 4
5 6 8

2 7 1
3 0 4
5 6 8

2 7 1
3 4 0
5 6 8

2 7 0
3 4 1
5 6 8

2 0 7
3 4 1
5 6 8

0 2 7
3 4 1
5 6 8

3 2 7
0 4 1
5 6 8

3 2 7
5 4 1
0 6 8

3 2 7
5 4 1
6 0 8

3 2 7
5 0 1
6 4 8

3 2 7
0 5 1
6 4 8

0 2 7
3 5 1
6 4 8

2 0 7
3 5 1
6 4 8

2 7 0
3 5 1
```

```
6 4 8

2 7 1
3 5 0
6 4 8

2 7 1
3 0 5
6 4 8

2 0 1
3 7 5
6 4 8

0 2 1
3 7 5
6 4 8

3 2 1
0 7 5
6 4 8

3 2 1
7 0 5
6 4 8

3 0 1
7 2 5
6 4 8

3 1 0
7 2 5
6 4 8

3 1 5
7 2 0
6 4 8

3 1 5
7 0 2
6 4 8

3 1 5
0 7 2
6 4 8

0 1 5
3 7 2
6 4 8

1 0 5
3 7 2
6 4 8

1 7 5
3 0 2
6 4 8
```

```
1 7 5
3 2 0
6 4 8

1 7 0
3 2 5
6 4 8

1 0 7
3 2 5
6 4 8

1 2 7
3 0 5
6 4 8

1 2 7
3 4 5
6 0 8

1 2 7
3 4 5
6 8 0

1 2 7
3 4 0
6 8 5

1 2 0
3 4 7
6 8 5

1 0 2
3 4 7
6 8 5

1 4 2
3 0 7
6 8 5

1 4 2
3 7 0
6 8 5

1 4 2
3 7 5
6 8 0

1 4 2
3 7 5
6 0 8

1 4 2
3 0 5
6 7 8

1 0 2
3 4 5
```

```
6 7 8

0 1 2
3 4 5
6 7 8
Number of steps taken: 67
Elapsed time: 0.03761482238769531
```