Lab 3: 8 Puzzle Single Player Game (A* Algorithm)

In [1]:

```
from time import time
from queue import PriorityQueue
import math
```

In [2]:

```
class Puzzle:
    goal_state=[1,2,3,8,0,4,7,6,5]
    heuristic=None
    evaluation_function=None
    needs_hueristic=False
    num_of_instances=0
    def __init__(self,state,parent,action,path_cost,needs_hueristic=False):
        self.parent=parent
        self.state=state
        self.action=action
        if parent:
            self.path_cost = parent.path_cost + path_cost
        else:
            self.path_cost = path_cost
        if needs_hueristic:
            self.needs_hueristic=True
            self.generate_heuristic()
            self.evaluation_function=self.heuristic+self.path_cost
        Puzzle.num_of_instances+=1

    def __str__(self):
        return str(self.state[0:3])+'\n'+str(self.state[3:6])+'\n'+str(self.state[6

    def generate_heuristic(self):
        self.heuristic=0
        for num in range(1,9):
            distance=abs(self.state.index(num) - self.goal_state.index(num))
            i=int(distance/3)
            j=int(distance%3)
            self.heuristic=self.heuristic+i+j

    def goal_test(self):
        if self.state == self.goal_state:
            return True
        return False

    @staticmethod
    def find_legal_actions(i,j):
        legal_action = ['U', 'D', 'L', 'R']
        if i == 0:  # up is disable
            legal_action.remove('U')
        elif i == 2:  # down is disable
            legal_action.remove('D')
        if j == 0:
            legal_action.remove('L')
        elif j == 2:
            legal_action.remove('R')
        return legal_action

    def generate_child(self):
        children=[]
        x = self.state.index(0)
        i = int(x / 3)
        j = int(x % 3)
        legal_actions=self.find_legal_actions(i,j)

        for action in legal_actions:
            new_state = self.state.copy()
            if action == 'U':
                new_state[x], new_state[x-3] = new_state[x-3], new_state[x]
            elif action == 'D':
                new_state[x], new_state[x+3] = new_state[x+3], new_state[x]
            elif action == 'L':
                new_state[x], new_state[x-1] = new_state[x-1], new_state[x]
            elif action == 'R':
                new_state[x], new_state[x+1] = new_state[x+1], new_state[x]
            children.append(Puzzle(new_state,self,action,1,self.needs_hueristic))
        return children

    def find_solution(self):
        solution = []
        solution.append(self.action)
        path = self
        while path.parent != None:
            path = path.parent
            solution.append(path.action)
        solution = solution[:-1]
        solution.reverse()
        return solution
```

In [3]:

```
def Astar_search(initial_state):
    count=0
    explored=[]
    start_node=Puzzle(initial_state,None,None,0,True)
    q = PriorityQueue()
    q.put((start_node.evaluation_function,count,start_node))

    while not q.empty():
        node=q.get()
        node=node[2]
        explored.append(node.state)
        if node.goal_test():
            return node.find_solution()

        children=node.generate_child()
        for child in children:
            if child.state not in explored:
                count += 1
                q.put((child.evaluation_function,count,child))
    return
```

In [4]:

```
#Start executing the 8-puzzle with setting up the initial state
#Here we have considered 3 initial state intitalized using state variable
state=[[1, 3, 4,
        8, 6, 2,
        7, 0, 5],

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

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

for i in range(0,3):
    Puzzle.num_of_instances = 0
    t0 = time()
    astar = Astar_search(state[i])
    t1 = time() - t0
    print('A*:',astar)
    print('space:', Puzzle.num_of_instances)
    print('time:', t1)
    print()
print('------------------------------------------')
```

```
A*: ['U', 'R', 'U', 'L', 'D']
space: 16
time: 0.0009305477142333984

A*: ['U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
space: 42
time: 0.0019366741180419922

A*: ['R', 'U', 'L', 'U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
space: 95
time: 0.004247426986694336

------------------------------------------
```