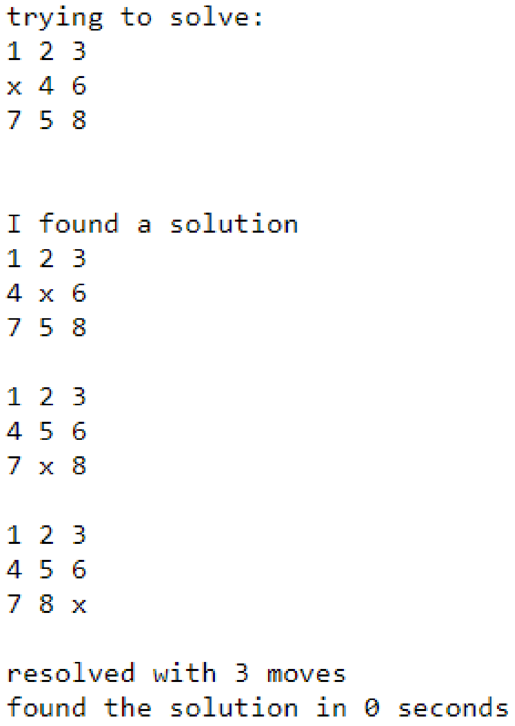
Aim: Write a program to solve 8 puzzle problem using A* algorithm.

Code:

```
from random import choice
from heapq import heappush, heappop ,heapify
from random import shuffle
import time

class Solver:
  def __init__(self, initial_state=None):
    self.initial_state = State(initial_state)
    self.goal = range(1, 9)
  def _rebuildPath(self, end):
    path = [end]
    state = end.parent
    while state.parent:
      path.append(state)
      state = state.parent
    return path
  def solve(self):
    openset = PriorityQueue()
    openset.add(self.initial_state)
    closed = set()
    moves = 0
    print 'trying to solve:'
    print openset.peek(), '\n\n'
    start = time.time()
    while openset:
      current = openset.poll()
      if current.values[:-1] == self.goal:
        end = time.time()
        print 'I found a solution'
        path = self._rebuildPath(current)
        for state in reversed(path):
          print state
          print
        print 'resolved with% d moves' % len(path)
        print 'found the solution in% 2.f seconds' % float(end - start)
        break
      moves += 1
      for state in current.possible_moves(moves):
        if state not in closed:
          openset.add(state)
      closed.add(current)
    else:
      print 'I could not solve it!'

class State:
  def __init__(self, values, moves=0, parent=None):
    self.values = values
    self.moves = moves
    self.parent = parent
    self.goal = range(1, 9)
  def possible_moves(self, moves):
    i = self.values.index(0)
    if i in [3, 4, 5, 6, 7, 8]:
      new_board = self.values[:]
      new_board[i], new_board[i - 3] = new_board[i - 3], new_board[i]
      yield State(new_board, moves, self)
    if i in [1, 2, 4, 5, 7, 8]:
      new_board = self.values[:]
      new_board[i], new_board[i - 1] = new_board[i - 1], new_board[i]
      yield State(new_board, moves, self)
    if i in [0, 1, 3, 4, 6, 7]:
      new_board = self.values[:]
      new_board[i], new_board[i + 1] = new_board[i + 1], new_board[i]
      yield State(new_board, moves, self)
    if i in [0, 1, 2, 3, 4, 5]:
      new_board = self.values[:]
      new_board[i], new_board[i + 3] = new_board[i + 3], new_board[i]
      yield State(new_board, moves, self)
  def score(self):
    return self._h() + self._g()
  def _h(self):
    return sum([1 if self.values[i] != self.goal[i] else 0 for i in xrange(8)])
  def _g(self):
    return self.moves
  def __cmp__(self, other):
    return self.values == other.values
  def __eq__(self, other):
    return self.__cmp__(other)
  def __hash__(self):
    return hash(str(self.values))
  def __lt__(self, other):
    return self.score() < other.score()
  def __str__(self):
    return '\n'.join([str(self.values[:3]),
        str(self.values[3:6]),
        str(self.values[6:9])]).replace('[', ").replace(']', ").replace(',', ").replace('0', 'x')

class PriorityQueue:
  def __init__(self):
    self.pq = []
  def add(self, item):
    heappush(self.pq, item)
  def poll(self):
    return heappop(self.pq)
  def peek(self):
    return self.pq[0]
  def remove(self, item):
    value = self.pq.remove(item)
    heapify(self.pq)
    return value is not None
  def __len__(self):
    return len(self.pq)
```

```
puzzle = range(9)

shuffle(puzzle)

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

solver = Solver(puzzle)

solver.solve()
```

Output:



```
trying to solve:
1 2 3
x 4 6
7 5 8


I found a solution
1 2 3
4 x 6
7 5 8

1 2 3
4 5 6
7 x 8

1 2 3
4 5 6
7 8 x

resolved with 3 moves
found the solution in 0 seconds
```