Name: Soham Devdatta Dhodapkar
ID: 801075881

Project 1: Solving 8-puzzle using A* search algorithm

8-Puzzle problem:

8-puzzle problem consists of an area divided into a grid, which is a 3x3 for the 8-puzzle problem. On each square tile in the grid is a tile (digit), except for one square which is a blank tile (here denoted by 0). The aim of this game is to slide the tiles one move at a time and achieve a goal state, which is the final configuration of the grid.

How to solve this? → A* algorithm:

A* algorithm is used widely in search, pathfinding and graph traversal. Using A* we can find paths between 2 nodes in a graph using heuristic functions which gives us a cost to traverse the path. Based on these costs, the algorithm will select the best possible path from A to B, or for finding a node N.
A* uses evaluation function $f(n) = g(n) + h(n)$ where for a node n, $g(n)$ denotes the path cost from start node to node n and $h(n)$ is the heuristic value (based on the heuristic method).

Possible moves:

The blank tile (here 0) can be moved by swapping it with the tile in four directions; UP, DOWN, LEFT, RIGHT. These moves keep on going until the goal configuration is reached. We can solve this using blind search or heuristic based methods.

Heuristics:

Heuristics are the rules we use to evaluate to a certain value, meaning that these rules will be used to calculate the path cost or whatever metric we want to calculate. We are using two heuristic search techniques to solve the 8-puzzle problem:

- **Misplaced tiles heuristic:** Based on the number of tiles that are misplaced from the goal configuration
- **Manhattan distance heuristic:** This calculates the distance between two points which is measured along the axes and at right angles.

This implementation aims to use both these heuristics with the A* algorithm.
Below are the steps that the program follows:

1. User gives inputs – Initial configuration and goal configuration
2. Check if current state is the goal state, if not then proceed further and add the node from the fringe to closed list.

3. Check for achievable states and check if the state has been previously generated, count these nodes.
4. Keep on looping until the goal state is reached or the fringe is empty.

Program structure:

8puzzle.py

1. main () will take the input from user for initial state and the goal state. Also, will ask which heuristic method to use to evaluate. It will then call the appropriate heuristic method, execute that, transfer control back to main and call the aStarSearch() function. After returning, will print the appropriate result and generated and expanded nodes.
2. __init__(), __str__(), __eq__(), __ne()__ and __hash()__ functions are standard defined functions in python which check for input errors, validations and object evaluations.
3. misplaced_tiles() will take the goal state as the argument, calculate this heuristic from the initial state and return the sum.
4. manhattan() will take the goal state as the argument, calculate this heuristic from the initial state and return the sum.
5. achievable_states() function will check for the next feasible states, put it in list and return them.
6. aStarSearch () will implement the A* algorithm in which we have maintained a fringe list, closed list and check with the achievable states.

Source code:

```
import re
import sys
import copy as cp

hueristics_list=[]
generated=[]

class eight_puzzle():
    #Init function will compile a regular exp where \d is a digit followed by a \s tha
t is a whitespace & input verification
    def __init__(self, str):
        regex = re.compile("(\d)\s(\d)\s(\d)\s(\d)\s(\d)\s(\d)\s(\d)\s(\d)\s(\d)") #
        result = regex.match(str)
        if result is not None:
            x = result.groups()
            self.state = [[int(x[0]), int(x[1]), int(x[2])],
                          [int(x[3]), int(x[4]), int(x[5])],
                          [int(x[6]), int(x[7]), int(x[8])]]
        else:
            print("Input error!")
```

```
#To return a string rep of the input matrix
def __str__(self):
    x = ''
    for i in range (0,3):
        for j in range(0,3):
            x += str(self.state[i][j]) + ' '
    return x

#Doing equality checks with eq and ne
def __eq__(self, other):
    if type(other) is type(self):
        return self.__dict__ == other.__dict__
    return False

def __ne__(self, other):
    return not self.__eq__(other)

#For performing operations on a set
def __hash__(self):
    uid = 0
    mult = 1
    for i in range(0,3):
        for j in range(0,3):
            uid += self.state[i][j] * mult
            mult *= 9
    return uid

#Function for implementing misplaced tiles
def misplaced_tiles(self, goal):
    sum = 0
    for i in range(0, 3):
        for j in range(0, 3):
            if (self.state[i][j] != goal.state[i][j]):
                sum += 1
    return sum

#Function for implementing Manhattan Distance
def manhattan(self, goal):
    sum = 0
    for i in range(0, 3):
        for j in range(0, 3):
            tile = self.state[i][j]
            for m in range(0, 3):
                for n in range(0, 3):
                    if tile == goal.state[m][n]:
                        sum += abs(i-m) + abs(j+n)
    return sum
```

```
    #Checking feasible next moves from a particular state
    def achievable_states(self):
        list = []
        idx = self.get_tile_zero()
        x = idx[0]
        y = idx[1]
        if x > 0:
            r = cp.deepcopy(self)
            r.state[y][x] = r.state[y][x-1]
            r.state[y][x-1] = 0
            list.append((r,'r'))
        if x < 2:
            l = cp.deepcopy(self)
            l.state[y][x] = l.state[y][x+1]
            l.state[y][x+1] = 0
            list.append((l,'l'))
        if y > 0:
            d = cp.deepcopy(self)
            d.state[y][x] = d.state[y-1][x]
            d.state[y-1][x] = 0
            list.append((d,'d'))
        if y < 2:
            u = cp.deepcopy(self)
            u.state[y][x] = u.state[y+1][x]
            u.state[y+1][x] = 0
            list.append((u,'u'))
        return list

    def get_tile_zero(self):
        for i in range(0, 3):
            for j in range(0, 3):
                if self.state[i][j] == 0:
                    x = j
                    y = i
        return (x,y)

    #A* algorithm -
- Storing nodes in a fringe, closed list, expanding nodes from the neighbour
    def aStarSearch(self, goal, heuristic, output):
        closed_list = set()
        fringe_list = set([self])
        prev = {}
        sum=0
        gOfn = {self : 0}
        fOfn = {self : gOfn[self] + heuristic(self,goal)}

        while (len(fringe_list) != 0):
```

```
            current = None
            for node in fringe_list:
                if current is None or f0fn[node] < f0fn[current]:
                    current = node
            if current == goal:
                return output(self, prev, current, heuristic)

            fringe_list.remove(current)
            closed_list.add(current)
            sum=0
            for n in current.achievable_states():
                sum+=1
                neighbour = n[0]
                if neighbour in closed_list:
                    continue
                temp_g0fn = g0fn[current] + 1

                if neighbour not in fringe_list or temp_g0fn < g0fn[neighbour]:
                    prev[neighbour] = (current, n[1])
                    g0fn[neighbour] = temp_g0fn
                    h=heuristic(neighbour,goal)
                    f0fn[neighbour] = g0fn[neighbour] + h
                    if neighbour not in fringe_list:
                        fringe_list.add(neighbour)
            generated.append(sum)
        return "empty"

    #State transition using selected heuristic
    def transition(self, prev, current_node, heuristic):
        goal = current_node
        return self.action_transition(prev, current_node, goal, heuristic)

    #Checking for a goal
    def action_transition(self, prev, current_node, goal, heuristic):
        delimiter = "\n"
        if current_node == goal:
            delimiter = ""
        if current_node in prev:
            p = self.action_transition(prev, prev[current_node][0], goal, heuristic)
            p += str(current_node) + delimiter
            hueristics_list.append(heuristic(current_node, goal))
            return p
        else:
            return str(current_node) + delimiter


# Main Function -- take inputs & call functions
def main():
```

```
    start = []
    print ("\nINITIAL CONFIGURATION")
    print ("Enter values for \n\n 1X1|1X2|1X3 \n ----------- \n 2X1|2X2|2X3 \n -------
---- \n 3X1|3X2|3X3 \n\n Please enter 0 as the blank tile ")
    user_input = input("Enter value for 1X1: ")
    start.append(user_input)
    user_input = input("Enter value for 1X2: ")
    start.append(user_input)
    user_input = input("Enter value for 1X3: ")
    start.append(user_input)
    user_input = input("Enter value for 2X1: ")
    start.append(user_input)
    user_input = input("Enter value for 2X2: ")
    start.append(user_input)
    user_input = input("Enter value for 2X3: ")
    start.append(user_input)
    user_input = input("Enter value for 3X1: ")
    start.append(user_input)
    user_input = input("Enter value for 3X2: ")
    start.append(user_input)
    user_input = input("Enter value for 3X3: ")
    start.append(user_input)

    initial_input = (' '.join(map(str, start)))

    end = []
    print ("\nGOAL CONFIGURATION")
    print ("Enter values for \n\n 1X1|1X2|1X3 \n ----------- \n 2X1|2X2|2X3 \n -------
---- \n 3X1|3X2|3X3 \n\n Please enter 0 as the blank tile ")
    user_input = input("Enter value for 1x1: ")
    end.append(user_input)
    user_input = input("Enter value for 1x2: ")
    end.append(user_input)
    user_input = input("Enter value for 1x3: ")
    end.append(user_input)
    user_input = input("Enter value for 2x1: ")
    end.append(user_input)
    user_input = input("Enter value for 2x2: ")
    end.append(user_input)
    user_input = input("Enter value for 2x3: ")
    end.append(user_input)
    user_input = input("Enter value for 3x1: ")
    end.append(user_input)
    user_input = input("Enter value for 3x2: ")
    end.append(user_input)
    user_input = input("Enter value for 3x3: ")
    end.append(user_input)
```

```
    goal_input = ' '.join(map(str, end))

        print("Choose your hueristic: \n")
        print ("1.Misplaced Tiles")
        print ("2.Manhattan distance \n")

        h = input("Enter your choice: ")
        if h == 1:
            heuristic = eight_puzzle.misplaced_tiles
        elif h == 2:
            heuristic = eight_puzzle.manhattan

        output = eight_puzzle.transition

        initial = eight_puzzle(initial_input)
        goal = eight_puzzle(goal_input)

        result = initial.aStarSearch(goal, heuristic, output)
        count=0
        g=1
        j=0
        res=''
        sum=0
        for i in result:
        if i==' ':
            count+=1
        if(count==3):
            res+='\n'
            count=0
        elif i=='\n':
            res+="\n"
            print
            g+=1
            j+=1
        else:
            res=res+i
        for i in generated:
        sum+=i
    print (res)
        print ("Nodes Generated = "),sum
        print ("Nodes Expanded = "),j

if __name__ == '__main__':
    main()
```

Outputs:

2 test cases taken from sample cases from project document. Test case 3 is taken from Homework 1

TEST CASE 1:

```
(base) CCI0NSFD58AWS:IS_Project1 sdhodapk$ python 8puzzle.py

INITIAL CONFIGURATION
Enter values for

 1X1|1X2|1X3
 ------------------
 2X1|2X2|2X3
 ------------------
 3X1|3X2|3X3

 Please enter 0 as the blank tile
Enter value for 1X1: 1
Enter value for 1X2: 2
Enter value for 1X3: 3
Enter value for 2X1: 7
Enter value for 2X2: 4
Enter value for 2X3: 5
Enter value for 3X1: 6
Enter value for 3X2: 8
Enter value for 3X3: 0

GOAL CONFIGURATION
Enter values for

 1X1|1X2|1X3
 -----------
 2X1|2X2|2X3
 -----------
 3X1|3X2|3X3

 Please enter 0 as the blank tile
Enter value for 1x1: 1
Enter value for 1x2: 2
Enter value for 1x3: 3
Enter value for 2x1: 8
Enter value for 2x2: 6
```

Enter value for 2x3: 4
Enter value for 3x1: 7
Enter value for 3x2: 5
Enter value for 3x3: 0
Choose your hueristic:

1.Misplaced Tiles
2.Manhattan distance

Enter your choice: 1

1 2 3
7 4 5
6 8 0

1 2 3
7 4 0
6 8 5

1 2 3
7 0 4
6 8 5

1 2 3
7 8 4
6 0 5

1 2 3
7 8 4
0 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4

7 6 5

1 2 3
8 6 4
7 0 5

1 2 3
8 6 4
7 5 0

Nodes Generated = 60
Nodes Expanded = 8

(base) CCI0NSFD58AWS:IS_Project1 sdhodapk$ python 8puzzle.py

INITIAL CONFIGURATION
Enter values for

1X1|1X2|1X3

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1X1: 1
Enter value for 1X2: 2
Enter value for 1X3: 3
Enter value for 2X1: 7
Enter value for 2X2: 4
Enter value for 2X3: 5
Enter value for 3X1: 6
Enter value for 3X2: 8
Enter value for 3X3: 0

GOAL CONFIGURATION
Enter values for

1X1|1X2|1X3

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1x1: 1
Enter value for 1x2: 2
Enter value for 1x3: 3
Enter value for 2x1: 8
Enter value for 2x2: 6
Enter value for 2x3: 4
Enter value for 3x1: 7
Enter value for 3x2: 5
Enter value for 3x3: 0
Choose your hueristic:

1.Misplaced Tiles
2.Manhattan distance

Enter your choice: 2

1 2 3
7 4 5
6 8 0

1 2 3
7 4 0
6 8 5

1 2 3
7 0 4
6 8 5

1 2 3
7 8 4
6 0 5

1 2 3
7 8 4
0 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5

1 2 3
8 6 4
7 0 5

1 2 3
8 6 4
7 5 0

Nodes Generated = 57
Nodes Expanded = 8

TEST CASE 2:

(base) CCI0NSFD58AWS:IS_Project1 sdhodapk$ python 8puzzle.py

INITIAL CONFIGURATION
Enter values for

```
 1X1|1X2|1X3
 -----------
 2X1|2X2|2X3
 -----------
 3X1|3X2|3X3
```

Please enter 0 as the blank tile
Enter value for 1X1: 2
Enter value for 1X2: 8
Enter value for 1X3: 1
Enter value for 2X1: 3
Enter value for 2X2: 4
Enter value for 2X3: 6
Enter value for 3X1: 7
Enter value for 3X2: 5
Enter value for 3X3: 0

GOAL CONFIGURATION
Enter values for

1X1|1X2|1X3

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1x1: 3
Enter value for 1x2: 2
Enter value for 1x3: 1
Enter value for 2x1: 8
Enter value for 2x2: 0
Enter value for 2x3: 4
Enter value for 3x1: 7
Enter value for 3x2: 5
Enter value for 3x3: 6
Choose your hueristic:

1.Misplaced Tiles
2.Manhattan distance

Enter your choice: 1

2 8 1
3 4 6
7 5 0

2 8 1
3 4 0
7 5 6

2 8 1
3 0 4
7 5 6

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

0 2 1
3 8 4
7 5 6

3 2 1
0 8 4
7 5 6

3 2 1
8 0 4
7 5 6

Nodes Generated =  20
Nodes Expanded =  6
(base) CCI0NSFD58AWS:IS_Project1 sdhodapk$ python 8puzzle.py

INITIAL CONFIGURATION
Enter values for

 1X1|1X2|1X3
 -----------
 2X1|2X2|2X3
 -----------
 3X1|3X2|3X3

 Please enter 0 as the blank tile
Enter value for 1X1: 2
Enter value for 1X2: 8
Enter value for 1X3: 1
Enter value for 2X1: 3
Enter value for 2X2: 4
Enter value for 2X3: 6
Enter value for 3X1: 7
Enter value for 3X2: 5
Enter value for 3X3: 0

GOAL CONFIGURATION
Enter values for

 1X1|1X2|1X3
```

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1x1: 3
Enter value for 1x2: 2
Enter value for 1x3: 1
Enter value for 2x1: 8
Enter value for 2x2: 0
Enter value for 2x3: 4
Enter value for 3x1: 7
Enter value for 3x2: 5
Enter value for 3x3: 6
Choose your hueristic:

1.Misplaced Tiles
2.Manhattan distance

Enter your choice: 2

2 8 1
3 4 6
7 5 0

2 8 1
3 4 0
7 5 6

2 8 1
3 0 4
7 5 6

2 0 1
3 8 4
7 5 6

0 2 1

3 8 4
7 5 6

3 2 1
0 8 4
7 5 6

3 2 1
8 0 4
7 5 6

Nodes Generated = 27
Nodes Expanded = 6

TEST CASE 3:

(base) CCI0NSFD58AWS:IS_Project1 sdhodapk$ python 8puzzle.py

INITIAL CONFIGURATION
Enter values for

1X1|1X2|1X3

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1X1: 4
Enter value for 1X2: 1
Enter value for 1X3: 3
Enter value for 2X1: 0
Enter value for 2X2: 2
Enter value for 2X3: 6
Enter value for 3X1: 7
Enter value for 3X2: 5
Enter value for 3X3: 8

GOAL CONFIGURATION
Enter values for

1X1|1X2|1X3

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1x1: 1
Enter value for 1x2: 2
Enter value for 1x3: 3
Enter value for 2x1: 4
Enter value for 2x2: 5
Enter value for 2x3: 6
Enter value for 3x1: 7
Enter value for 3x2: 8
Enter value for 3x3: 0
Choose your hueristic:

1.Misplaced Tiles
2.Manhattan distance

Enter your choice: 1

4 1 3
0 2 6
7 5 8

0 1 3
4 2 6
7 5 8

1 0 3
4 2 6
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6

7 0 8

1 2 3
4 5 6
7 8 0

Nodes Generated = 15
Nodes Expanded = 5

(base) CCI0NSFD58AWS:IS_Project1 sdhodapk$ python 8puzzle.py

INITIAL CONFIGURATION
Enter values for

1X1|1X2|1X3

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1X1: 4
Enter value for 1X2: 1
Enter value for 1X3: 3
Enter value for 2X1: 0
Enter value for 2X2: 2
Enter value for 2X3: 6
Enter value for 3X1: 7
Enter value for 3X2: 5
Enter value for 3X3: 8

GOAL CONFIGURATION
Enter values for

1X1|1X2|1X3

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1x1: 1
Enter value for 1x2: 2
Enter value for 1x3: 3

Enter value for 2x1: 4
Enter value for 2x2: 5
Enter value for 2x3: 6
Enter value for 3x1: 7
Enter value for 3x2: 8
Enter value for 3x3: 0
Choose your hueristic:

1.Misplaced Tiles
2.Manhattan distance

Enter your choice: 2

4 1 3
0 2 6
7 5 8

0 1 3
4 2 6
7 5 8

1 0 3
4 2 6
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

Nodes Generated = 33
Nodes Expanded = 5

TEST CASE 4:

(base) CCI0NSFD58AWS:IS_Project1 sdhodapk$ python 8puzzle.py

INITIAL CONFIGURATION
Enter values for

```
 1X1|1X2|1X3
 -----------
 2X1|2X2|2X3
 -----------
 3X1|3X2|3X3

 Please enter 0 as the blank tile
Enter value for 1X1: 1
Enter value for 1X2: 2
Enter value for 1X3: 3
Enter value for 2X1: 0
Enter value for 2X2: 4
Enter value for 2X3: 6
Enter value for 3X1: 7
Enter value for 3X2: 5
Enter value for 3X3: 8
```

GOAL CONFIGURATION
Enter values for

```
 1X1|1X2|1X3
 -----------
 2X1|2X2|2X3
 -----------
 3X1|3X2|3X3

 Please enter 0 as the blank tile
Enter value for 1x1: 1
Enter value for 1x2: 2
Enter value for 1x3: 3
Enter value for 2x1: 4
Enter value for 2x2: 5
Enter value for 2x3: 6
Enter value for 3x1: 7
```

Enter value for 3x2: 8
Enter value for 3x3: 0
Choose your hueristic:

1.Misplaced Tiles
2.Manhattan distance

Enter your choice: 1

1 2 3
0 4 6
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

Nodes Generated = 10
Nodes Expanded = 3
(base) CCI0NSFD58AWS:IS_Project1 sdhodapk$ python 8puzzle.py

INITIAL CONFIGURATION
Enter values for

1X1|1X2|1X3

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1X1: 1
Enter value for 1X2: 2
Enter value for 1X3: 3

Enter value for 2X1: 0
Enter value for 2X2: 4
Enter value for 2X3: 6
Enter value for 3X1: 7
Enter value for 3X2: 5
Enter value for 3X3: 8

GOAL CONFIGURATION
Enter values for

1X1|1X2|1X3

2X1|2X2|2X3

3X1|3X2|3X3

Please enter 0 as the blank tile
Enter value for 1x1: 1
Enter value for 1x2: 2
Enter value for 1x3: 3
Enter value for 2x1: 4
Enter value for 2x2: 5
Enter value for 2x3: 6
Enter value for 3x1: 7
Enter value for 3x2: 8
Enter value for 3x3: 0
Choose your hueristic:

1.Misplaced Tiles
2.Manhattan distance

Enter your choice: 2

1 2 3
0 4 6
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3

4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

Nodes Generated = 10
Nodes Expanded = 3

Table:

Test case	Initial state	Goal state	Misplaced tiles	Manhattan distance
1	1 2 3 7 4 5 6 8 0	1 2 3 8 6 4 7 5 0	Nodes Generated = 60 Nodes Expanded = 8	Nodes Generated = 57 Nodes Expanded = 8
2	2 8 1 3 4 6 7 5 0	3 2 1 8 0 4 7 5 6	Nodes Generated = 27 Nodes Expanded = 6	Nodes Generated = 20 Nodes Expanded = 6
3	4 1 3 0 2 6 7 5 8	1 2 3 4 5 6 7 8 0	Nodes Generated = 15 Nodes Expanded = 5	Nodes Generated = 33 Nodes Expanded = 5
4	1 2 3 0 4 6 7 5 8	1 2 3 4 5 6 7 8 0	Nodes Generated = 10 Nodes Expanded = 3	Nodes Generated = 10 Nodes Expanded = 3