Sanchit Gupta

Project 1- Planning for Autonomous Robots
8-Puzzle solver using Breadth First Search Algorithm

The code for solving the 8-Puzzle by sliding the tile labelled "0" and arranging the digits in ascending order, while putting the 0 in the end is written in Python. The name of the file is **8_Puzzle_solver.py**

The goal node has the following shape:

1	2	3
4	5	6
7	8	0

Before running the program, please ensure that Numpy and the OS module is installed. Numpy allows you to create and modify array in python while OS module helps you perform operating system dependent functionality within Python like creating a file, deleting an already made file using its path.

After these packages have been installed, you can go ahead and run the program. Once running, the program will ask the user to input the problem puzzle, the code should look like this:

```
C:\Users\sanch\Desktop\Python_Programs\venv\Scripts\python.exe C:/Users/sanch/Desktop/Python_Programs/8_Puzzle_solver.py
Please enter number from 0-8, no number should be repeated or be out of this range
Enter the 1 number: 1
Enter the 2 number: 2
Enter the 3 number:
```

1	2	3
4	5	6
7	8	9

When it says enter the 2 number, it means this number in the puzzle structure, similar is true for all the other digits

Once all the numbers have been put and they are all within the range of [0-8], the program will start running. If you enter any number which is not in this range, the program will quit with an error message and you will have to re-start putting the number.

The program will then check if any number, which is given to the puzzle is repeated or not, if a number has been repeated, even then an error message will be printed, and the program will quit out as this type of input is not allowed. This is done using the *"def check_correct_input(l)"* function

If the input is acceptable, the program will then check if the input puzzle is solvable or not. In the 8-puzzle game there are certain input cases where the final goal node is never reached, when you give such an input, the program will detect it and print a warning message that the following set might not

have a solution, but it will still run and check all the nodes. In this condition, all the nodes are explored in approximately 45 min and "Goal Node could not be reached" error is printed out. To check if the input state has a solution or not, the program has the function called *"def check_solvable(g)",* in this function, what we do is convert the input state into a single row array and then count the number of times the digits occur in reverse order that they should occur in the goal node. if that counter returns an even value then the puzzle is solvable and if it is odd then it isn't.

Once you have an input which is solvable and is correct, the code starts to run, and "exploring Nodes" statement is printed on your console. This means that the code is running and trying to find the configuration which matches the goal node which is already hard-coded.

The data structure used to store the puzzle information and the parent information is Node, which has been defined as a class in the python file.

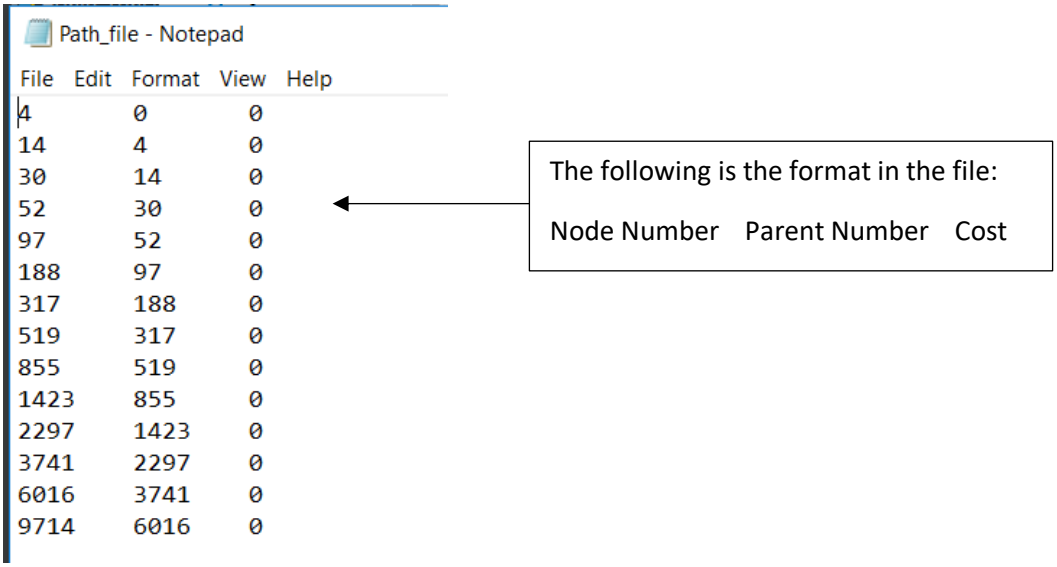
It contains the following attributes:

1. Node Number: each node when it is formed is assigned a unique number
2. Data: the actual puzzle
3. Parent: Parent Node
4. act: the action which was used to create this node
5. Cost: the cost to create this node

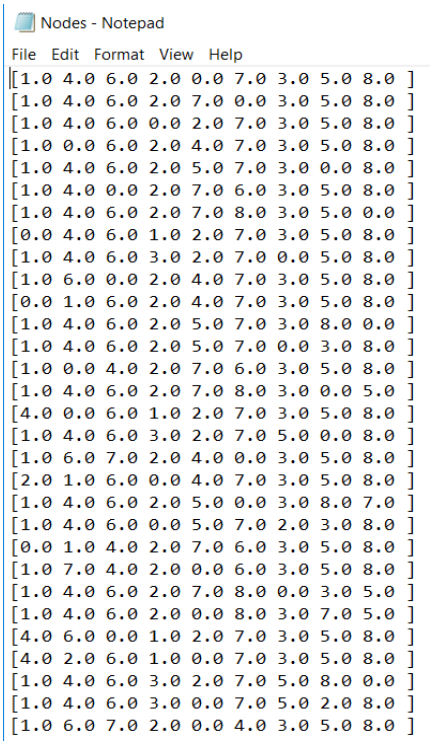
As soon as a node is formed, which matches the goal node, the loop breaks, and 4 different functions are called.

- The first function is used to print the path which should be travelled to reach the goal state from the input puzzle stage, with their corresponding moves.

- The second function is used to print the same path into a text file, this function checks if there is a file name already by the name – "Path_file.txt", is there is it deletes it and creates a new Path_file.txt with new information.



- The third function is used to print in a text file, the nodes it visited/created while finding the goal node. This file also finds the file named “Nodes.txt”, if it is there it deletes it and then creates it with new information.



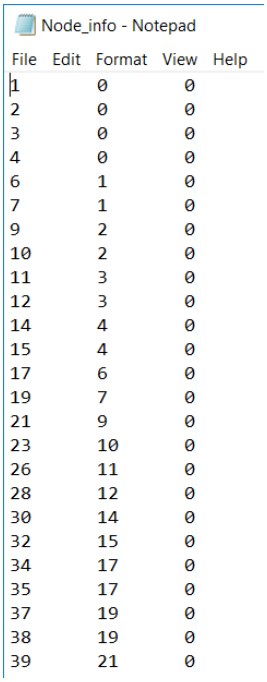
Nodes - Notepad

File Edit Format View Help

```
[1.0 4.0 6.0 2.0 0.0 7.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 2.0 7.0 0.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 0.0 2.0 7.0 3.0 5.0 8.0 ]
[1.0 0.0 6.0 2.0 4.0 7.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 2.0 5.0 7.0 3.0 0.0 8.0 ]
[1.0 4.0 0.0 2.0 7.0 6.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 2.0 7.0 8.0 3.0 5.0 0.0 ]
[0.0 4.0 6.0 1.0 2.0 7.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 3.0 2.0 7.0 0.0 5.0 8.0 ]
[1.0 6.0 0.0 2.0 4.0 7.0 3.0 5.0 8.0 ]
[0.0 1.0 6.0 2.0 4.0 7.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 2.0 5.0 7.0 3.0 8.0 0.0 ]
[1.0 4.0 6.0 2.0 5.0 7.0 0.0 3.0 8.0 ]
[1.0 0.0 4.0 2.0 7.0 6.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 2.0 7.0 8.0 3.0 0.0 5.0 ]
[4.0 0.0 6.0 1.0 2.0 7.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 3.0 2.0 7.0 5.0 0.0 8.0 ]
[1.0 6.0 7.0 2.0 4.0 0.0 3.0 5.0 8.0 ]
[2.0 1.0 6.0 0.0 4.0 7.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 2.0 5.0 0.0 3.0 8.0 7.0 ]
[1.0 4.0 6.0 0.0 5.0 7.0 2.0 3.0 8.0 ]
[0.0 1.0 4.0 2.0 7.0 6.0 3.0 5.0 8.0 ]
[1.0 7.0 4.0 2.0 0.0 6.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 2.0 7.0 8.0 0.0 3.0 5.0 ]
[1.0 4.0 6.0 2.0 0.0 8.0 3.0 7.0 5.0 ]
[4.0 6.0 0.0 1.0 2.0 7.0 3.0 5.0 8.0 ]
[4.0 2.0 6.0 1.0 0.0 7.0 3.0 5.0 8.0 ]
[1.0 4.0 6.0 3.0 2.0 7.0 5.0 8.0 0.0 ]
[1.0 4.0 6.0 3.0 0.0 7.0 5.0 2.0 8.0 ]
[1.0 6.0 7.0 2.0 0.0 4.0 3.0 5.0 8.0 ]
```

Here each row represents a node with its 9 elements

- The fourth function is used to print the node information in a text file, it prints the Node Number, Node Parent number and the cost for each node travelled by the program. It is saved by the name “Nodes_info.txt”



Node_info - Notepad

File Edit Format View Help

```
1       0       0
2       0       0
3       0       0
4       0       0
6       1       0
7       1       0
9       2       0
10      2       0
11      3       0
12      3       0
14      4       0
15      4       0
17      6       0
19      7       0
21      9       0
23      10      0
26      11      0
28      12      0
30      14      0
32      15      0
34      17      0
35      17      0
37      19      0
38      19      0
39      21      0
```

The following is the format in the file:

Node Number Parent Number Cost

Code is pasted here as well:

```
""" 8 Puzzle BFS algorithm
 Input the unsolved puzzle and the program
 solves it and creates 3 txt files with solutions"""

import numpy as np  # Used to store the digits in an array
import os  # Used to delete the file created by previous running of the program


class Node:
    def __init__(self, node_no, data, parent, id, cost):
        self.data = data
        self.parent = parent
        self.id = id
        self.node_no = node_no
        self.cost = cost


def get_initial():
    print("Please enter number from 0-8, no number should be repeated or be out of this range")
    initial_state = np.zeros(9)
    for i in range(9):
        states = int(input("Enter the " + str(i + 1) + " number: "))
        if states < 0 or states > 8:
            print("Please only enter states which are [0-8], run code again")
            exit(0)
        else:
            initial_state[i] = np.array(states)
    return np.reshape(initial_state, (3, 3))


def find_index(puzzle):
    i, j = np.where(puzzle == 0)
    i = int(i)
    j = int(j)
    return i, j


def move_left(data):
    i, j = find_index(data)
    if j == 0:
        return None
    else:
        temp_arr = np.copy(data)
        temp = temp_arr[i, j-1]
        temp_arr[i, j] = temp
        temp_arr[i, j-1] = 0
        return temp_arr


def move_right(data):
    i, j = find_index(data)
    if j == 2:
        return None
    else:
        temp_arr = np.copy(data)
        temp = temp_arr[i, j+1]
        temp_arr[i, j] = temp
        temp_arr[i, j+1] = 0
        return temp_arr


def move_up(data):
    i, j = find_index(data)
    if i == 0:
        return None
    else:
        temp_arr = np.copy(data)
        temp = temp_arr[i-1, j]
        temp_arr[i, j] = temp
        temp_arr[i-1, j] = 0
        return temp_arr


def move_down(data):
    i, j = find_index(data)
    if i == 2:
        return None
    else:
```

```
        temp_arr = np.copy(data)
        temp = temp_arr[i+1, j]
        temp_arr[i, j] = temp
        temp_arr[i+1, j] = 0
        return temp_arr


def move_tile(action, data):
    if action == 'up':
        return move_up(data)
    if action == 'down':
        return move_down(data)
    if action == 'left':
        return move_left(data)
    if action == 'right':
        return move_right(data)
    else:
        return None


def print_states(list_final):  # To print the final states on the console
    print("printing final solution")
    for l in list_final:
        print("Move : " + str(l.id) + "\n" + "Result : " + "\n" + str(l.data) + "\t" + "node number:" +
str(l.node_no))


def write_path(path_formed):  # To write the final path in the text file
    if os.path.exists("Path_file.txt"):
        os.remove("Path_file.txt")

    f = open("Path_file.txt", "a")
    for node in path_formed:
        if node.parent is not None:
            f.write(str(node.node_no) + "\t" + str(node.parent.node_no) + "\t" + str(node.cost)+"\n")
    f.close()


def write_node_explored(explored):  # To write all the nodes explored by the program
    if os.path.exists("Nodes.txt"):
        os.remove("Nodes.txt")

    f = open("Nodes.txt", "a")
    for element in explored:
        for i in range(len(element)):
            f.write(str(element[i])+" ")
        f.write("\n")
    f.close()


def write_node_info(visited):  # To write all the info about the nodes explored by the program
    if os.path.exists("Node_info.txt"):
        os.remove("Node_info.txt")

    f = open("Node_info.txt", "a")
    for n in visited:
        if n.parent is not None:
            f.write(str(n.node_no) + "\t" + str(n.parent.node_no) + "\t" + str(n.cost)+"\n")
    f.close()


def path(node):  # To find the path from the goal node to the starting node
    p = []  # Empty list
    p.append(node)
    parent_node = node.parent
    while parent_node is not None:
        p.append(parent_node)
        parent_node = parent_node.parent
    return list(reversed(p))


def exploring_nodes(node):
    print("Exploring Nodes")
    actions = ["down", "up", "left", "right"]
    goal_node = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 0]])
    node_q = [node]
    final_nodes = []
    visited = []
    final_nodes.append(node_q[0].data.tolist())  # Only writing data of nodes in seen
    node_counter = 0  # To define a unique ID to all the nodes formed
```

```
    while node_q:
        current_root = node_q.pop(0)  # Pop the element 0 from the list
        if current_root.data.tolist() == goal_node.tolist():
            print("Goal reached")
            return current_root, final_nodes, visited

        for move in actions:
            temp_data = move_tile(move, current_root.data)
            if temp_data is not None:
                node_counter += 1
                child_node = Node(node_counter, np.array(temp_data), current_root, move, 0)  # Create a child
node

                if child_node.data.tolist() not in final_nodes:  # Add the child node data in final node list
                    node_q.append(child_node)
                    final_nodes.append(child_node.data.tolist())
                    visited.append(child_node)
                    if child_node.data.tolist() == goal_node.tolist():
                        print("Goal reached")
                        return child_node, final_nodes, visited
    return None, None, None  # return statement if the goal node is not reached


def check_correct_input(l):
    array = np.reshape(l, 9)
    for i in range(9):
        counter_appear = 0
        f = array[i]
        for j in range(9):
            if f == array[j]:
                counter_appear += 1
        if counter_appear >= 2:
            print("invalid input, same number entered 2 times")
            exit(0)


def check_solvable(g):
    arr = np.reshape(g, 9)
    counter_states = 0
    for i in range(9):
        if not arr[i] == 0:
            check_elem = arr[i]
            for x in range(i+1, 9):
                if check_elem < arr[x] or arr[x] == 0:
                    continue
                else:
                    counter_states += 1
    if counter_states % 2 == 0:
        print("The puzzle is solvable, generating path")
    else:
        print("The puzzle is insolvable, still creating nodes")


# Final Running of the Code

# uncomment the line below to run it for a fixed data input and comment the line below it
#k = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 0]])
k = get_initial()

check_correct_input(k)
check_solvable(k)

root = Node(0, k, None, None, 0)

# BFS implementation call
goal, s, v = exploring_nodes(root)

if goal is None and s is None and v is None:
    print("Goal State could not be reached, Sorry")
else:
    # Print and write the final output
    print_states(path(goal))
    write_path(path(goal))
    write_node_explored(s)
    write_node_info(v)
```