# 8 Puzzle problem

**Github repository**: https://github.com/sivabalachandran/AI-Assignments

## Assumptions:

- All puzzles are solvable.

## Code Layout:

Code is structured to achieve reusability and modularity by providing following classes and files. The goal is defined in the puzzle board so as to not be maintained in multiple files.

**PuzzleBoard.py:**

It represents the puzzle board class and provides utility functions to return possible board configurations that can be achieved by moving the empty tile up, down, left and right.

**Util.py:**

Provides following functionality.

1. A priority queue that acts as a min-heap for UCS.
2. A utility function that gathers and validates user input.
3. A print function that prints the path traversed.

The get initial configuration function validates the input to make sure the input is a number between 0-8. Else it restarts the program and asks the user to input data again.

*Sample Input and Output*:

```
/usr/bin/python3.9
/home/standard/Documents/ksu/CS7375-ArtificialIntelligence/Assignments/UCS.py
Please input the board data - Number must be between 0 - 8
Enter data for tile 1:- a



==== Please enter a number between 0 -8. Lets start from the beginning. ====
Please input the board data - Number must be between 0 - 8
Enter data for tile 1:- 10
```

```
==== Please enter a number between 0 -8. Lets start from the beginning. ====
Please input the board data - Number must be between 0 - 8
Enter data for tile 1:-
```

**BFS.py:**

Solves the given board using the BFS algorithm. As the name indicates it processes boards(nodes) at a given tree level before moving onto the next level. It achieves this by processing child nodes that are reachable from the current board and adds it to a queue. The boards are processed from the head of the queue. If the popped board is the goal state, it stops and short circuits the problem, else it retrieves the possible child boards and adds them to the end of the queue.

The design also stores a list of visited states. This helps in optimizing the solution by not exploring it again.

*Sample Input and Output:*

```
Please input the board data- Number must be between 0 - 8
Enter data for tile 1:- 1
Enter data for tile 2:- 3
Enter data for tile 3:- 4
Enter data for tile 4:- 8
Enter data for tile 5:- 0
Enter data for tile 6:- 5
Enter data for tile 7:- 7
Enter data for tile 8:- 2
Enter data for tile 9:- 6
========= Solution ==========
Starting Configuration :-
 [[1 3 4]
 [8 0 5]
 [7 2 6]]
move DOWN from above configuration
Achieved board :-
 [[1 3 4]
 [8 2 5]
 [7 0 6]]
-------------------------------------
move RIGHT from above configuration
Achieved board :-
```

```
 [[1 3 4]
 [8 2 5]
 [7 6 0]]
--------------------------------------
move UP from above configuration
Achieved board :-
 [[1 3 4]
 [8 2 0]
 [7 6 5]]
--------------------------------------
move UP from above configuration
Achieved board :-
 [[1 3 0]
 [8 2 4]
 [7 6 5]]
--------------------------------------
move LEFT from above configuration
Achieved board :-
 [[1 0 3]
 [8 2 4]
 [7 6 5]]
--------------------------------------
move DOWN from above configuration
Achieved board :-
 [[1 2 3]
 [8 0 4]
 [7 6 5]]
--------------------------------------
--- Actions taken --
DOWN
RIGHT
UP
UP
LEFT
DOWN
```

**DFS:**

Solves the given board using the DFS algorithm. As the name indicates it processes a node and all of its children before moving up the tree to drill down further. The design uses a stack and adds  children of a given node to the stack. The boards are processed from the top of the stack. The algorithm pops the top element to begin with. If the popped board is the goal state, it stops and short circuits the problem, else it retrieves the possible child boards and adds them to the stack. Before adding the algorithm also check if it is already visited or if it is the goal state. The solution employs depth limiting DFS search to avoid running for a long time.

The design also stores a list of visited states. This helps in optimizing the solution by not exploring it again and again.

*Sample Input and Output:*

```
/usr/bin/python3.9
/home/standard/Documents/ksu/CS7375-ArtificialIntelligence/Assignments/DFS.py
Please input the board data- Number must be between 0 - 8
Enter data for tile 1:- 1
Enter data for tile 2:- 3
Enter data for tile 3:- 4
Enter data for tile 4:- 8
Enter data for tile 5:- 0
Enter data for tile 6:- 5
Enter data for tile 7:- 7
Enter data for tile 8:- 2
Enter data for tile 9:- 6
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
could not find path even after depth 5, so skipping the node and dipping into stack
========= Solution ==========
Starting Configuration :-
 [[1 3 4]
 [8 0 5]
 [7 2 6]]
move DOWN from above configuration
Achieved board :-
 [[1 3 4]
 [8 2 5]
 [7 0 6]]
-------------------------------------
move RIGHT from above configuration
Achieved board :-
 [[1 3 4]
 [8 2 5]
 [7 6 0]]
-------------------------------------
move UP from above configuration
Achieved board :-
 [[1 3 4]
 [8 2 0]
 [7 6 5]]
-------------------------------------
move UP from above configuration
Achieved board :-
 [[1 3 0]
 [8 2 4]
 [7 6 5]]
-------------------------------------
move LEFT from above configuration
```

```
Achieved board :-
 [[1 0 3]
 [8 2 4]
 [7 6 5]]
-------------------------------------
move DOWN from above configuration
Achieved board :-
 [[1 2 3]
 [8 0 4]
 [7 6 5]]
-------------------------------------
--- Actions taken --
DOWN
RIGHT
UP
UP
LEFT
DOWN

Process finished with exit code 0
```

**UCS:**

Solves the given board using the UCS algorithm. The design uses a Priority queue that acts as a min heap. The min heap sorts the elements based on the cost involved. The algorithm pops the elements with min cost. If the popped board is the goal state, it stops and short circuits the problem, else it retrieves the possible child boards and adds them to the min heap. Before adding the algorithm also check if it is already visited or if it is the goal state.

The design also stores a list of visited states. This helps in optimizing the solution by not exploring it again and again.

*Sample Input and Output:*

```
/usr/bin/python3.9
/home/standard/Documents/ksu/CS7375-ArtificialIntelligence/Assignments/UCS.py
Please input the board data- Number must be between 0 - 8
Enter data for tile 1:- 1
Enter data for tile 2:- 3
Enter data for tile 3:- 4
Enter data for tile 4:- 8
Enter data for tile 5:- 0
Enter data for tile 6:- 5
Enter data for tile 7:- 7
Enter data for tile 8:- 2
Enter data for tile 9:- 6
========= Solution ==========
Starting Configuration :-
 [[1 3 4]
 [8 0 5]
 [7 2 6]]
move DOWN from above configuration
Achieved board :-
 [[1 3 4]
 [8 2 5]
 [7 0 6]]
--------------------------------------
move RIGHT from above configuration
Achieved board :-
 [[1 3 4]
 [8 2 5]
 [7 6 0]]
--------------------------------------
move UP from above configuration
Achieved board :-
 [[1 3 4]
 [8 2 0]
 [7 6 5]]
--------------------------------------
move UP from above configuration
Achieved board :-
 [[1 3 0]
 [8 2 4]
 [7 6 5]]
--------------------------------------
move LEFT from above configuration
Achieved board :-
 [[1 0 3]
 [8 2 4]
 [7 6 5]]
--------------------------------------
move DOWN from above configuration
Achieved board :-
 [[1 2 3]
 [8 0 4]
 [7 6 5]]
```

```
--------------------------------------
--- Actions taken --
DOWN
RIGHT
UP
UP
LEFT
DOWN

Process finished with exit code 0
```