**Instructions to Run the 8-puzzle code**

Files:

1. Input.java

The file contains the main function, and all the common functions and parameters required by other files

1. BFS.java

The file contains the main logic of BFS algorithm

1. IDS.java

The file contains the main logic of IDS algorithm

1. AStar.java

The file contains the main logic of AStar algorithm. The two heuristic functions used are Incorrect Place and Manhattan Distance

1. Node.java

The node which is used for the search algorithms. It contains all the information of the state

Assumptions:

1. The tree can go to maximum depth of 10 (Change it to whatever you want in Input.java)
2. If the child of the node is the goal state, then the search will return the path from root to child
3. Number of states enqueued denotes the states that are added to queue/stack
4. User will enter the numbers row-wise in the range 1-8 without repeating any number and one blank space which denotes \*
5. The code is only for solving 8-puzzle problems (change the parameter PUZZLE\_SIZE if you want to change to 15-puzzle and so on)
6. User will always enter the correct number when asked about which algorithm or which heuristic should be used

Run:

1. Compile all the given files

javac \*.java

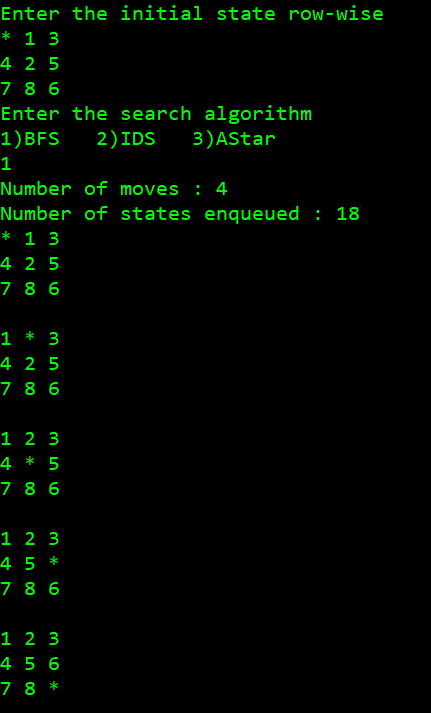
1. Run the Input.java which has the main function

java Input

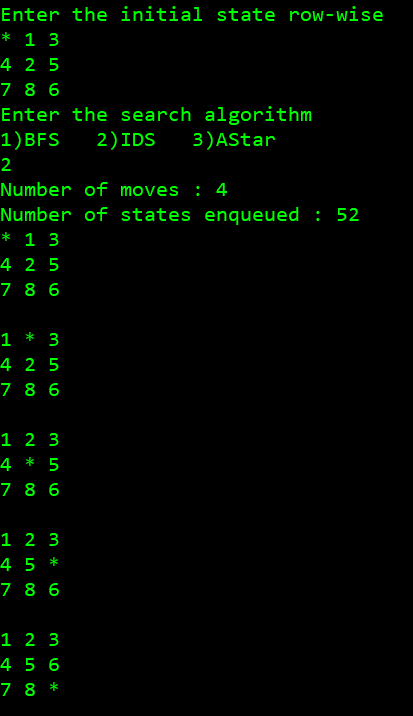
1. Enter the position of numbers as asked
2. Enter the search algorithm number you wish to perform
3. Enter the heuristic number if you want to perform A\*

**Screenshots of sample input and its corresponding output:**

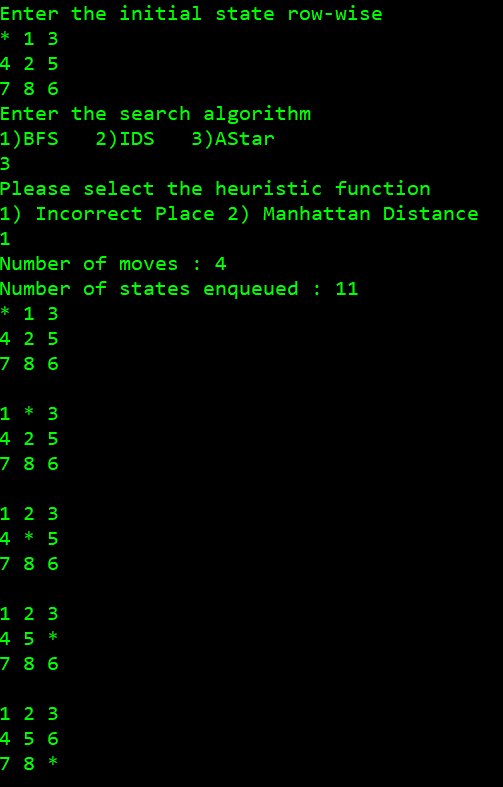
1. BFS



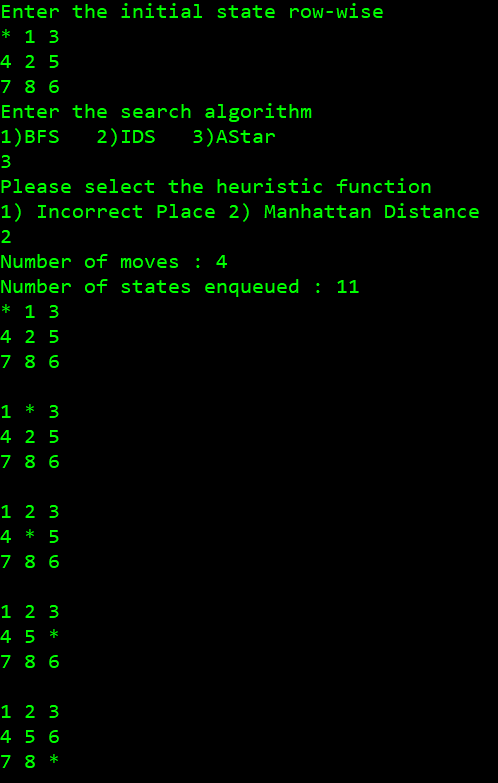
1. IDS



1. A\* (Incorrect Place Heuristic)



1. A\* (Manhattan Distance Heuristic)



**Comparative analysis of two heuristics:**

Both the heuristics are admissible. Wrong place heuristic is admissible because the numbers which are in wrong position must move at least once. Manhattan distance heuristic is admissible as the numbers cannot move is diagonal direction so to reach to its original position it has to travel vertically and horizontally

One of the ways to determine the goodness of the heuristic is the effective branching factor. The factor is close to 1 if the heuristic is well designed. Other method is to calculate the number of nodes generated by that search tree or the time it takes to find the goal state

In my code, I checked the time required by search algorithm for completing the search. For more complex problems, there was a failure, as the depth of the tree was more than 10 and if I tried to increase the depth, it took lot of time in finding the time. For not so complex problems, the time taken for heuristic with Incorrect Place was 1-2 milliseconds more compared to heuristic with Manhattan Distance

Also, according to some research done in the book ‘Artificial Intelligence’ by Russell and Norvig, it shows that the heuristic with Manhattan Distance is far more efficient than heuristic with Incorrect Place. This is logical too and can be explained

Consider for example, this state

\*12

543

678

Incorrect Place heuristic will have value of 8 which is max in this type of heuristic as all the tiles are in incorrect place, whereas Manhattan Distance will have value of 8 which is very low in this type of heuristic as every number is just 1 position away

Here, in this type of example, Incorrect Place heuristic will try to expand other states, but we know in just few steps this can be solved into the goal state. This can be achieved using Manhattan Distance heuristic