

# PROGRAMMING ASSIGNMENT – 1

CIS 579

FALL 2024

Vishvendra Reddy Bhoomidi

## Command to compile the program:

```
python eightpuzzle.py --search "A*" --initial "[[-,2,3],[1,4,5],[8,7,6]]" --goal "[[1,2,3],[8,-,4],[7,6,5]]" --heuristic manhattan
```

```
python eightpuzzle.py --search DFS --initial "[[-,2,3],[1,4,5],[8,7,6]]" --goal "[[1,2,3],[8,-,4],[7,6,5]]"
```

## COMPARISON TABLE:

ALGORITHM	HEURISTIC	PATH COST	DEPTH	TIME TAKEN	NODES EXPANDED
DFS	N/A	428	428	0.0074 seconds	435
BFS	N/A	6	6	0.0006 seconds	58
UCS	N/A	6	6	0.0006 seconds	58
GBS	Misplaced	6	6	0.0001 seconds	6
GBS	Manhattan	6	6	0.0001 seconds	6
A*	Misplaced	6	6	0.0001 seconds	6
A*	Manhattan	6	6	0.0001 seconds	6

## Search Algorithms:

Depth-First Search (DFS):

DFS explores nodes deeply before backtracking, resulting in high path cost and depth. It tends to explore more nodes than necessary, making it less efficient compared to other algorithms.

Breadth-First Search (BFS):

BFS explores nodes level by level, ensuring that the solution is found at the shallowest depth. However, it may explore many unnecessary nodes before finding the goal, leading to a high number of expanded nodes.

Uniform Cost Search (UCS):

UCS expands nodes based on the lowest path cost. In the case of the 8-puzzle, where path costs are uniform, UCS behaves similarly to BFS, resulting in the same path cost, depth, and number of nodes expanded.

### Greedy Best-First Search (GBS) with Misplaced Tile Heuristic:

GBS with the misplaced tile heuristic quickly finds solutions with very few nodes expanded. However, this heuristic does not guarantee the optimal path in general, although it performed efficiently in this case.

### Greedy Best-First Search (GBS) with Manhattan Distance Heuristic:

GBS with the Manhattan distance heuristic generally leads to more efficient solutions than the misplaced tiles heuristic. While it performed similarly in this puzzle, it would likely outperform it in larger puzzles.


### A\* Search with Misplaced Tile Heuristic:

A\* combines path cost and the misplaced tile heuristic to guarantee optimal solutions. It performs similarly to GBS for this puzzle but guarantees the optimal solution (least cost).

### A\* Search with Manhattan Distance Heuristic:

A\* with the Manhattan distance heuristic is highly efficient and guarantees optimal solutions by combining the accurate heuristic with a path cost, resulting in very few nodes expanded.

Output.txt:



The screenshot shows a code editor with several files open: 'eightpuzzle.py', 'search.py', 'ReadMe.md', 'output.txt' (selected), 'output\_DFS.txt', and 'util.py'. The 'output.txt' file contains a table comparing different search algorithms.

Search Algorithms Comparison Table:						
Algorithm	Heuristic	Path Cost	Depth	Time Taken	Nodes Expanded	
DFS	N/A	428	428	0.0074 sec	435	
BFS	N/A	6	6	0.0006 sec	58	
UCS	N/A	6	6	0.0006 sec	58	
GBS	misplaced	6	6	0.0001 sec	6	
GBS	manhattan	6	6	0.0001 sec	6	
A*	misplaced	6	6	0.0001 sec	6	
A*	manhattan	6	6	0.0001 sec	6	

Example output of output\_DFS.txt:

```

1 DFS Search Algorithm:
2 *****
3 Solution Path:
4 [State: [[0, 2, 3], [1, 4, 5], [8, 7, 6]] Action: down]
5 [State: [[0, 2, 3], [1, 4, 5], [8, 7, 6]] Action: right]
6 [State: [[2, 0, 3], [1, 4, 5], [8, 7, 6]] Action: down]
7 [State: [[2, 0, 3], [1, 4, 5], [8, 7, 6]] Action: right]
8 [State: [[2, 3, 0], [1, 4, 5], [8, 7, 6]] Action: down]
9 [State: [[2, 3, 5], [1, 4, 0], [8, 7, 6]] Action: down]
10 [State: [[2, 3, 5], [1, 4, 0], [8, 7, 6]] Action: left]
11 [State: [[2, 3, 5], [1, 0, 4], [8, 7, 6]] Action: up]
12 [State: [[2, 3, 5], [1, 0, 4], [8, 7, 6]] Action: down]
13 [State: [[2, 3, 5], [1, 0, 4], [8, 7, 6]] Action: left]
14 [State: [[2, 3, 5], [0, 1, 4], [8, 7, 6]] Action: up]
15 [State: [[2, 3, 5], [0, 1, 4], [8, 7, 6]] Action: down]
16 [State: [[2, 3, 5], [8, 1, 4], [0, 7, 6]] Action: right]
17 [State: [[2, 3, 5], [8, 1, 4], [7, 0, 6]] Action: up]
18 [State: [[2, 3, 5], [8, 1, 4], [7, 0, 6]] Action: right]
19 [State: [[2, 3, 5], [8, 1, 4], [7, 6, 0]] Action: up]
20 [State: [[2, 3, 5], [8, 1, 0], [7, 6, 4]] Action: up]
21 [State: [[2, 3, 5], [8, 1, 0], [7, 6, 4]] Action: left]
22 [State: [[2, 3, 5], [8, 0, 1], [7, 6, 4]] Action: up]
23 [State: [[2, 3, 5], [8, 0, 1], [7, 6, 4]] Action: down]
24 [State: [[2, 3, 5], [8, 0, 1], [7, 6, 4]] Action: left]
25 [State: [[2, 3, 5], [0, 8, 1], [7, 6, 4]] Action: up]
26 [State: [[2, 3, 5], [0, 8, 1], [7, 6, 4]] Action: down]
27 [State: [[1, 2, 3], [0, 7, 8], [6, 5, 4]] Action: up]
28 [State: [[1, 2, 3], [0, 7, 8], [6, 5, 4]] Action: right]
29 [State: [[1, 2, 3], [0, 7, 8], [6, 5, 4]] Action: right]
30 [State: [[1, 2, 3], [7, 0, 8], [6, 5, 4]] Action: up]
31 [State: [[1, 2, 3], [7, 0, 8], [6, 5, 4]] Action: down]
32 [State: [[1, 2, 3], [7, 0, 8], [6, 5, 4]] Action: right]
33 [State: [[1, 2, 3], [7, 8, 0], [6, 5, 4]] Action: up]
34 [State: [[1, 2, 3], [7, 8, 0], [6, 5, 4]] Action: down]
35 [State: [[1, 2, 3], [7, 8, 4], [6, 5, 0]] Action: left]
36 [State: [[1, 2, 3], [7, 8, 4], [6, 0, 5]] Action: up]
37 [State: [[1, 2, 3], [7, 8, 4], [6, 0, 5]] Action: left]
38 [State: [[1, 2, 3], [7, 8, 4], [0, 6, 5]] Action: up]
39 [State: [[1, 2, 3], [0, 8, 4], [7, 6, 5]] Action: up]
40 [State: [[1, 2, 3], [0, 8, 4], [7, 6, 5]] Action: right]
41 [State: [[1, 2, 3], [8, 0, 4], [7, 6, 5]] Action: Goal Reached]
42 Path Cost: 428
43 Depth: 428
44 Time Taken: 0.0074 seconds
45 Nodes Expanded: 435
46 *****
47
```

## References:

1. <https://docs.python.org/3/library/argparse.html>: The website teaches how to handle inputs
2. <https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/> : Details on how the Problem works
3. [https://www.w3schools.com/python/ref\\_func\\_eval.asp](https://www.w3schools.com/python/ref_func_eval.asp) and <https://docs.python.org/3/library/itertools.html>: For understanding some minute concepts.
4. <https://cs50.harvard.edu/ai/2024/notes/0/>: Gives knowledge about different Search Algorithms
5. PyCharm AI assistant was used to correct some grammatical errors during documentation and comments.