Assignment 1
Using Informed and Uninformed Search Algorithms to Solve 8-Puzzle

Rawan Ibrahim El-Sayed : 9082
Salma Yehia Ali : 9085
Mirolla Wael : 8602

1.Problem Overview:

This project implements and compares four search algorithms:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Iterative Deepening Depth-First Search (IDDFS)
- A*

to find a path from a given initial 8-puzzle configuration to the goal state.
The 8-puzzle consists of a 3×3 grid containing tiles numbered 1–8 and one blank space (represented by `0`). The goal is to reach the configuration “012345678” by sliding tiles horizontally or vertically into the blank space.

Each algorithm explores possible states of the puzzle (represented as strings of 9 digits) and attempts to find a sequence of valid moves that leads to the goal configuration.

2. Assumptions and Problem Representation

- The puzzle is represented as a string of 9 characters `"123045678"` . `'0'` represents the blank tile.
- Only up, down, left, and right moves are allowed.
- The goal state is fixed as `"012345678"`.
- Each move has an equal cost of 1.
- The helper functions `move_blank()`, `get_children()`, and `get_path()` are reused across all algorithms to generate child states, move the blank, and reconstruct the final path.

  Before searching, a solvability check (based on the inversion count) is performed to ensure that the given puzzle can reach the goal.

3. Data Structures Used

Algorithm	Data Structures	Purpose
BFS	`queue` (`collections.deque`)	FIFO structure to explore nodes level by level
DFS	`stack` (`deque` using `.pop()`)	LIFO structure to explore nodes deeply first
IDDFS	`stack`	Depth-limited DFS combined with iterative depth increase
A*	`priority queue` (`heapq`)	Expands nodes with the lowest total estimated cost f(n)=g(n)+h(n)

4. Algorithm Explanations

4.1 Breadth-First Search (BFS)

BFS explores the search space level by level.
It starts from the initial state and generates all possible child states before moving deeper.
Because BFS visits states in increasing order of depth, it guarantees the shortest (optimal) path to the goal when all moves have equal cost.

Data structure: FIFO queue (`popleft()` operation).
Termination: When the goal state is dequeued.
Complexity:

- Time: O(b^d)

- Space: O(b^d)
  where *b* is the branching factor and *d* is the depth of the goal.

Frontier: [Initial State]

Expand → Add children to queue → Explore next level

Advantages: Always finds the shortest path.

Disadvantages: High memory usage for deep puzzles.

4.2 Depth-First Search (DFS)

DFS explores as deep as possible along one branch before backtracking.
It uses a stack (LIFO) to keep track of states.
DFS is memory-efficient, but it may get stuck exploring a very deep or infinite branch, and it does not guarantee the shortest path.

Data structure: Stack (`pop()` operation).
Termination: When goal is popped from the stack.
Complexity:

- Time: O(b^m)
- Space: O(bm)
  where *m* is the maximum depth of the search tree.

Advantages: Low memory consumption.
Disadvantages: Can take a long path to the goal or fail if the goal is deep.

4.3 Iterative Deepening DFS (IDDFS)

IDDFS combines the space efficiency of DFS and the optimality of BFS.
It repeatedly performs a Depth-Limited DFS, increasing the depth limit by 1 each iteration until the goal is found.
This ensures that the first time the goal is reached, it is at the shallowest (optimal) depth.

Data structures: Stack for DFS; iteration over increasing depth limit.
Complexity:

- Time: O(b^d)
- Space: O(bd)

Advantages:

- Optimal like BFS
- Low memory like DFS

Disadvantages:

- Repeats work at shallow levels several times, slightly slower than BFS in practice.

4.4 A* Search

A* is an informed search algorithm that uses a heuristic to guide the exploration.
It selects the node with the lowest estimated total cost:

f(n)=g(n)+h(n)

where:

- g(n)) = cost from start to node *n*

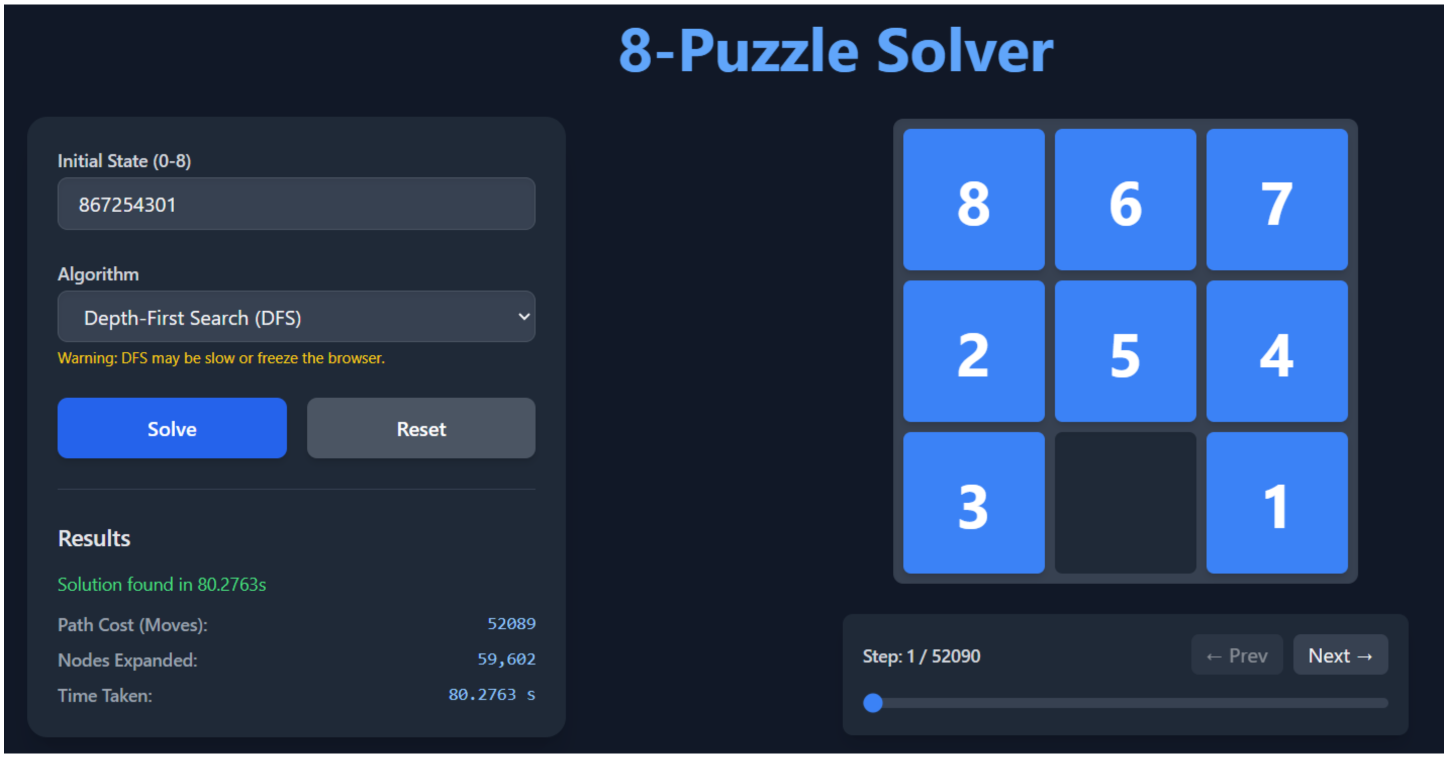
- h(n) = estimated cost from *n* to goal (heuristic)

  For the 8-puzzle, the common heuristic used is Manhattan Distance, which sums the distances of each tile from its goal position.

Data structure: Priority queue (min-heap) sorted by *f(n)*.
Complexity: Depends on heuristic quality; generally faster than uninformed searches.

Advantages: Fast and optimal when heuristic is admissible.
Disadvantages: Higher memory consumption than DFS or IDDFS.

5. Sample Runs and Results



Initial State (0-8)
867254301

Algorithm
Depth-First Search (DFS)

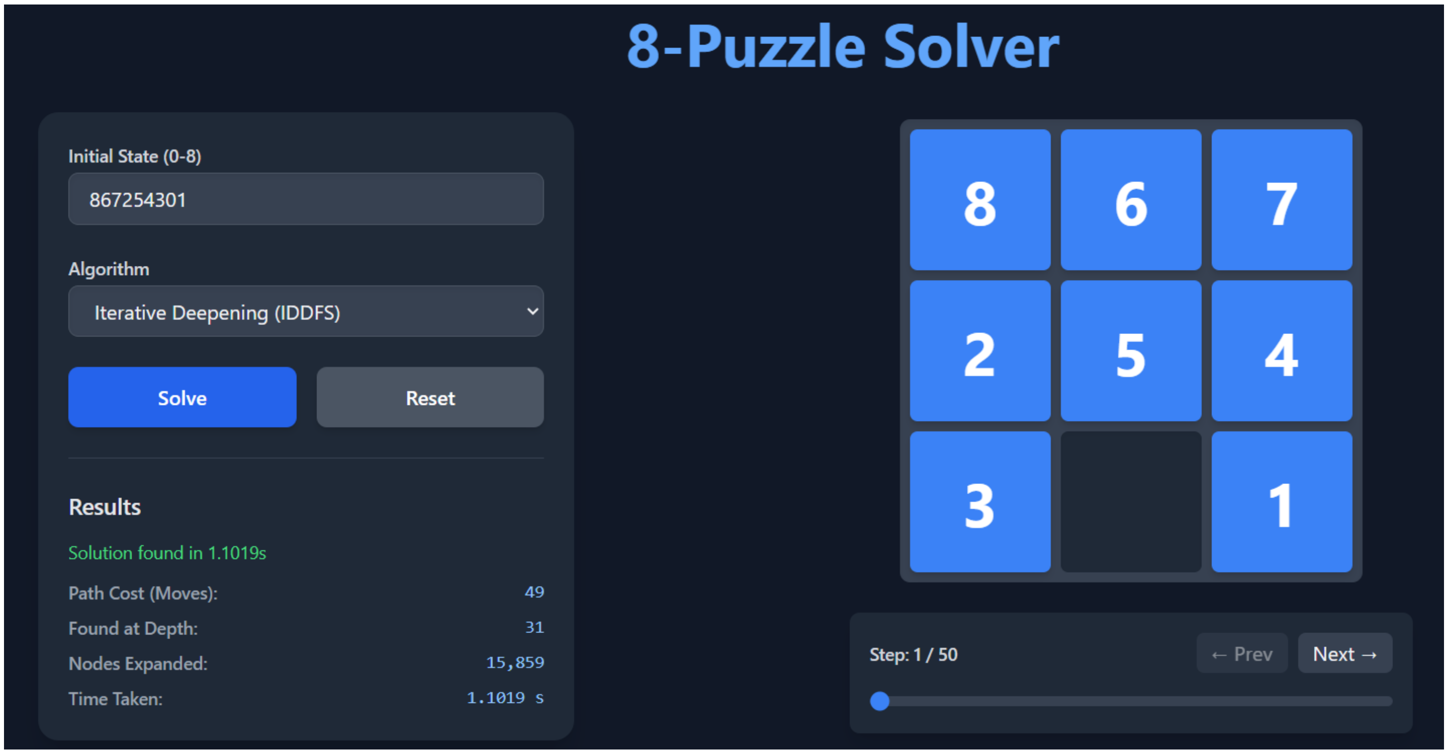
Warning: DFS may be slow or freeze the browser.

Results

Solution found in 80.2763s

Path Cost (Moves):	52089
Nodes Expanded:	59,602
Time Taken:	80.2763 s

Step: 1 / 52090



Initial State (0-8)
867254301

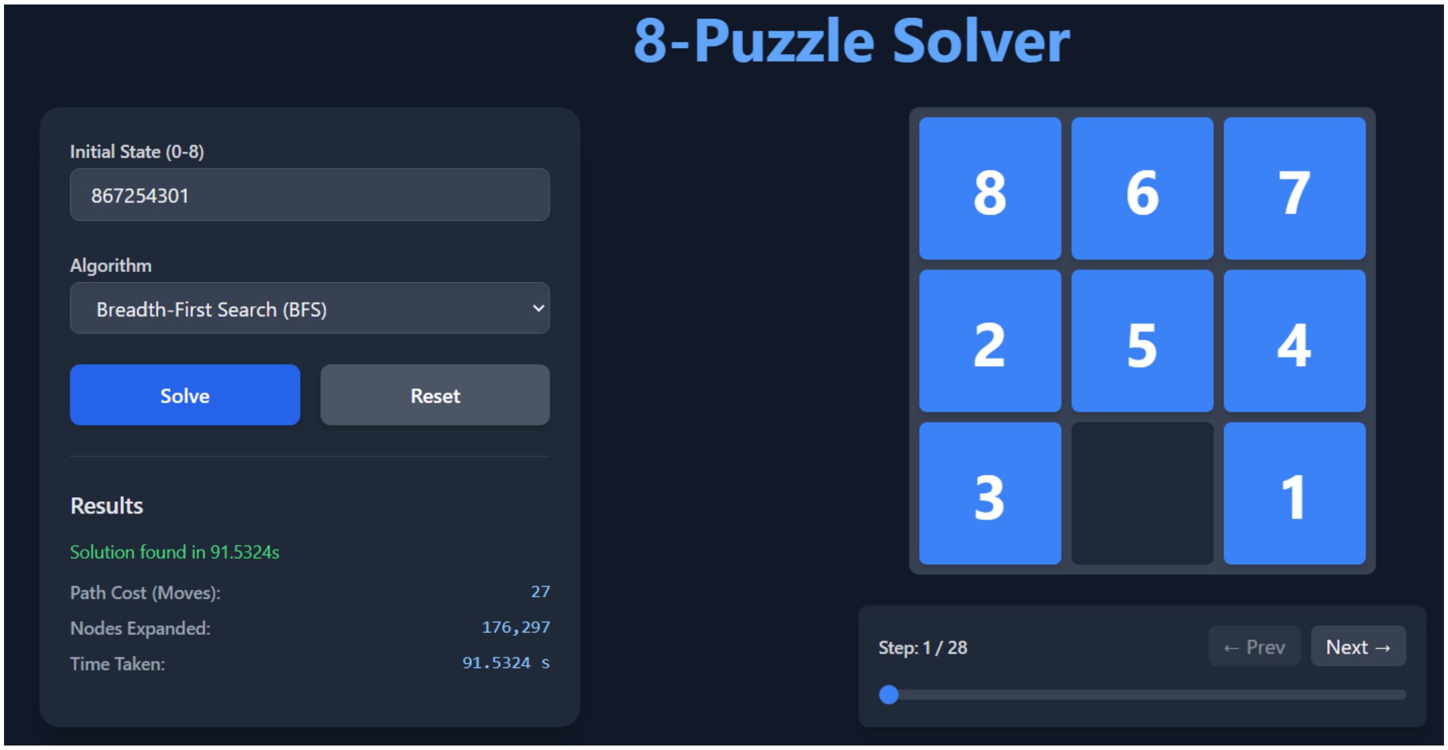
Algorithm
Iterative Deepening (IDDFS)

Results

Solution found in 1.1019s

Path Cost (Moves):	49
Found at Depth:	31
Nodes Expanded:	15,859
Time Taken:	1.1019 s

Step: 1 / 50



8-Puzzle Solver

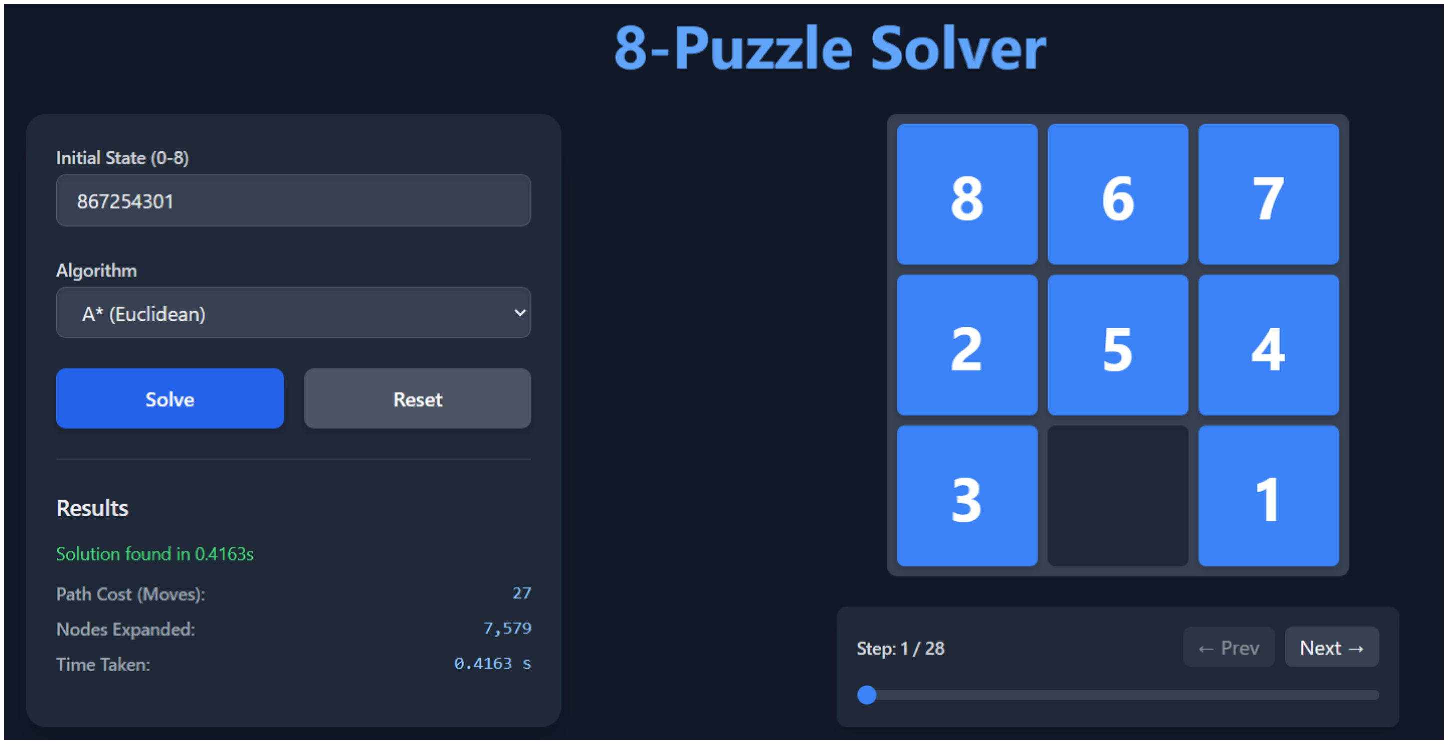
Initial State (0-8)
867254301

Algorithm
Breadth-First Search (BFS)

Results
Solution found in 91.5324s

```
Path Cost (Moves):      27
Nodes Expanded:    176,297
Time Taken:      91.5324 s
```

Step: 1 / 28



8-Puzzle Solver

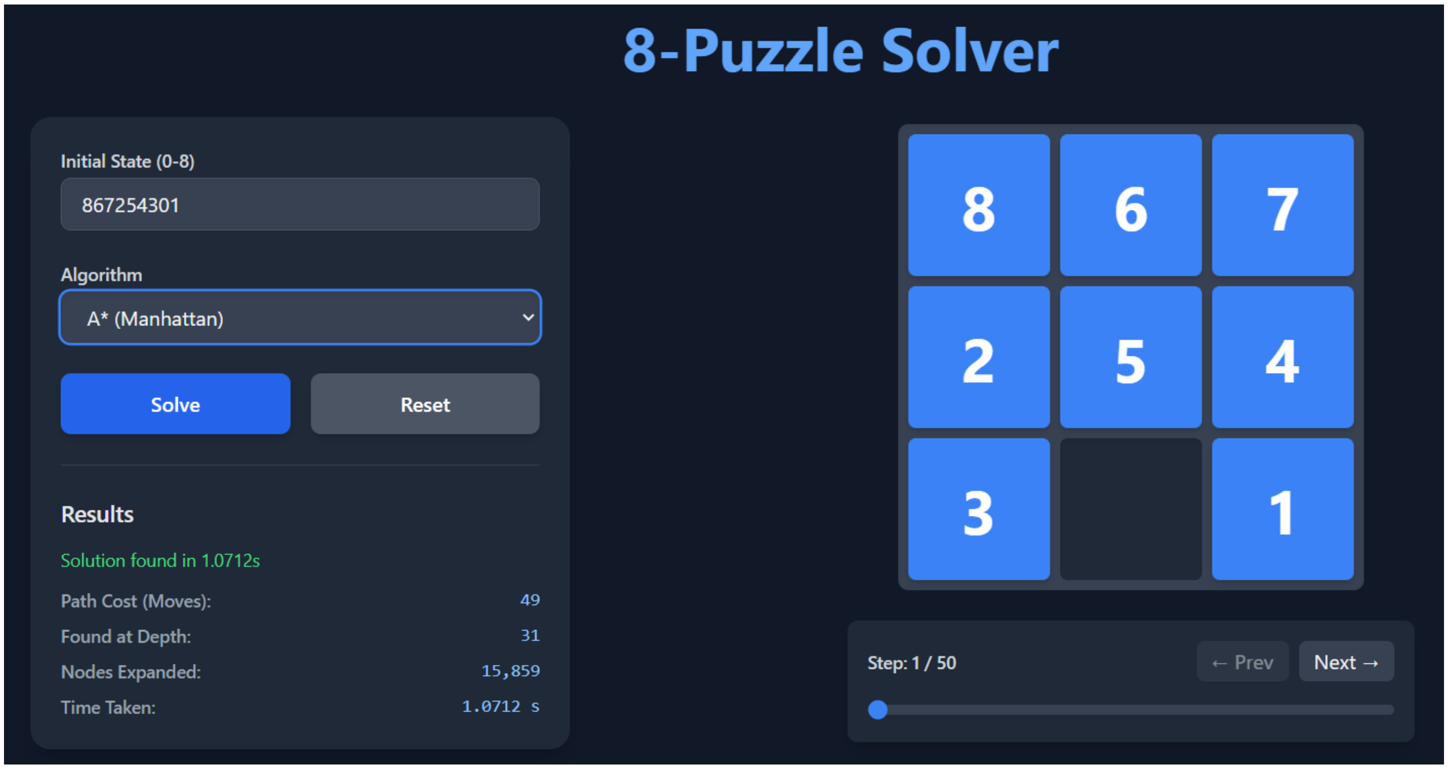
Initial State (0-8)
867254301

Algorithm
A* (Euclidean)

Results
Solution found in 0.4163s

```
Path Cost (Moves):      27
Nodes Expanded:      7,579
Time Taken:       0.4163 s
```

Step: 1 / 28



8-Puzzle Solver

Initial State (0-8)

867254301

Algorithm

A* (Manhattan)

Results

Solution found in 1.0712s

```
Path Cost (Moves):       49
Found at Depth:          31
Nodes Expanded:      15,859
Time Taken:        1.0712 s
```

Step: 1 / 50

6. Observations and Discussion

- BFS always finds the shortest path but may expand many unnecessary nodes for deeper puzzles.
- DFS consumes little memory but can explore large parts of the state space without progress and may return a non-optimal or wrong path.
- IDDFS balances both, it guarantees an optimal solution with very little memory overhead.
- A* is the most efficient when a good heuristic (Manhattan distance) is used, as it prioritizes promising states.

7. Conclusion

All four algorithms successfully explore the 8-puzzle search space, but their performance differs significantly:

- BFS: Reliable and optimal but memory-heavy.
- DFS: Simple but inefficient for deep puzzles.
- IDDFS: Best choice for memory-limited optimal search.
- A*: Fastest and most practical when a heuristic is available.

For this project, BFS and IDDFS provided the most accurate solutions, while DFS helped demonstrate search depth behavior.

A* provided the best trade-off between speed and optimality.