1. Introduction

Search algorithms are basic techniques used in Artificial Intelligence.

They help solve problems where a set of steps is needed to move

from a starting state to a goal state.

In this project, different search algorithms are implemented

to solve the 8-Puzzle problem.

The algorithms are compared based on solution quality,

number of explored states, and execution time.

---

2. Problem Description

The 8-Puzzle problem is a well-known AI problem.

It consists of a 3×3 grid that contains eight numbered tiles

and one empty space.

The tiles are moved by sliding them into the empty position.

The main goal is to reach the goal state shown below.

Initial State:

1 2 3

4 0 6

7 5 8

Goal State:

1 2 3

4 5 6

7 8 0

---

3. State Representation

Each puzzle state is represented as a one-dimensional array
with a size of 9 elements.
Number 0 represents the empty space.
New states are generated by moving the empty space
up, down, left, or right when the move is allowed.
Each state also keeps a reference to its parent state
to allow rebuilding the solution path.

---

4. Implemented Search Algorithms

4.1 Breadth First Search (BFS)

Breadth First Search explores the puzzle states level by level.

It always finds the shortest solution path when all moves

have the same cost.

However, BFS uses a large amount of memory because it stores

many explored states.

4.2 Depth First Search (DFS)

Depth First Search explores one path deeply before moving

to another path.

It uses less memory than BFS, but it does not guarantee finding the shortest solution

and may explore unnecessary paths.

4.3 A* Search (Manhattan Distance)

A* search uses both the actual path cost and a heuristic

to guide the search.

The Manhattan Distance heuristic calculates how far each tile

is from its correct position.

This heuristic is admissible, so A* always finds the optimal

solution while exploring fewer states than uninformed algorithms.

4.4 Hill Climbing

Hill Climbing is a local search algorithm based on a heuristic.

It always chooses the next state with the best heuristic value.

This algorithm is very fast and uses little memory,

but it does not guarantee reaching the goal.

It may get stuck in local optima or flat areas.

---

5. Heuristic Function

The Manhattan Distance heuristic calculates the total horizontal

and vertical distance of each tile from its goal position.

Using this heuristic helps reduce the number of explored states,

especially when using informed search algorithms like A*.

---

6. Performance Comparison

The following results were obtained using the same initial state:

Initial State:

1 2 3

4 0 6

7 5 8

Algorithm	Nodes Explored	Solution Steps	Time (ms)	Optimal
BFS	11	2	11	Yes
DFS	683	434	12	No
A* (Manhattan)	3	2	11	Yes
Hill Climbing	3	2	8	No

---

7. Conclusion

From the results, A* with Manhattan Distance gives the best performance.

It reaches the goal using the fewest explored states and the shortest path.

BFS also finds the correct solution but explores more states.

DFS and Hill Climbing can be faster, but they do not always give reliable or optimal solutions.