8-Puzzle Problem

Team Members:

1- Toka Fawzy Mohamed Elhwahi (C2)
Implement problem with A*

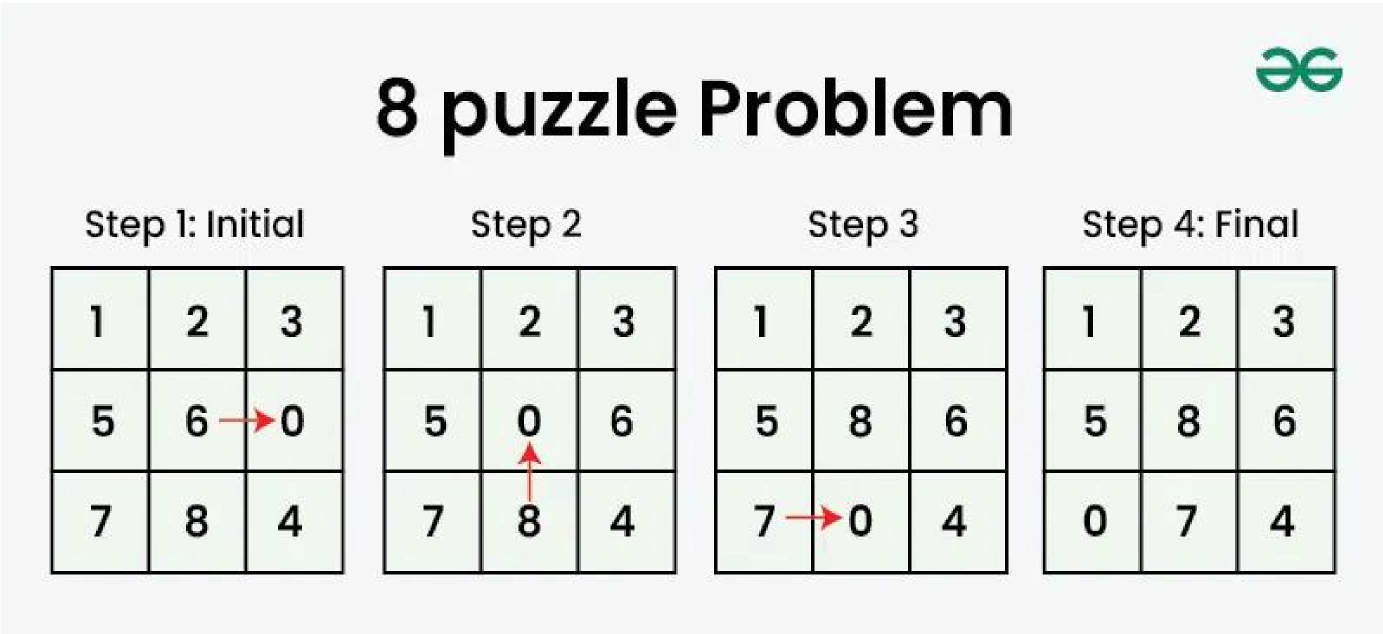
2- Mariam Ehab Ali Elkharrat (C5)
Implement problem with GBFS

3- Mariam Mohamed Faisal Nafee (C4)
Implement problem with IDS

4- Mennah Ragab Negm Elsayed Agwah (C4)
Implement problem with BFS

Problem Description

Given a 3×3 board with 8 tiles (each numbered from 1 to 8) and one empty space, **the objective** is to place the numbers to match the final configuration using the empty space. We can slide four adjacent tiles (left, right, above, and below) into the empty space.



We Will implement the problem with :

Breadth-first search(BFS):

BFS Overview

- **Structure:** Uses a **Queue** (FIFO) to track nodes.
- **Strategy:** Explores all neighbors at the current depth before moving to the next level.
- **Goal:** Returns the solution immediately upon reaching the target state.

Key Advantages:

- **Optimal Solution**: Guaranteed to find the fastest/shortest path because it explores moves level by level.
- **Completeness:** Guaranteed to find a solution if one exists.

Greedy Best-First Search (GBFS):

GBFS Overview

- **Strategy:** GBFS uses a heuristic function—often $h(n)$—to estimate the distance from the current node to the goal.
- **Structure:** It uses a Priority Queue to always expand the node that appears "closest" to the goal.
- **Efficiency:** It is generally faster than BFS because it focuses only on the most promising paths rather than exploring level-by-level.

Key Characteristics

- **Not Optimal:** Unlike BFS, GBFS may not find the shortest path; it only finds a path quickly.
- **Heuristic-Driven:** Its performance depends heavily on the accuracy of the heuristic (e.g., Misplaced Tiles or Manhattan Distance).
- **Completeness:** It can get stuck in infinite loops in a search space with cycles unless visited states are tracked.

Iterative Deepening Search (IDS):

IDS Overview

- **Strategy:** Combines the benefits of DFS (low memory) and BFS (guaranteed shortest path).
- **Mechanism:** It performs a Depth-Limited Search (DLS) repeatedly, increasing the depth limit by 1 in each iteration until the goal is found.
- **Logic:** It explores all nodes at depth 1, then depth 2, and so on, re-starting from the root each time.

Key Characteristics

- **Optimal:** Like BFS, it is guaranteed to find the shortest path (fewest moves).
- **Memory Efficient:** It only needs to store a single path at a time (linear space complexity), making it much better for memory than BFS.
- **Time Trade-off:** It repeats the work of earlier levels, but because most nodes are in the deepest level, this overhead is surprisingly small.

A Star Search (A*):

A* Overview

- **Strategy:** A* is an informed search algorithm that finds the shortest path by combining the strengths of BFS and GBFS.
- **Logic:** It uses the cost function $f(n) = g(n) + h(n)$ to decide which node to expand.
  - **g(n):** The actual cost (number of steps) from the start node to the current node.
  - **h(n):** The heuristic estimate of the cost from the current node to the goal.

- **Structure:** Like GBFS, it uses a **Priority Queue** to always expand the node with the lowest f(n) value.

Key Characteristics

- **Optimal and Complete:** A* is guaranteed to find the shortest path if the heuristic h(n) is "admissible" (meaning it never overestimates the actual cost).
- **Efficiency:** It is much more efficient than BFS because it avoids exploring paths that are clearly moving away from the goal.
- **Balance:** While BFS only cares about $g(n)$ and GBFS only cares about $h(n)$, A* balances both to ensure both speed and accuracy.

Execution Time results:

1. **IDS** will be the slowest but uses the least RAM.

2. **GBFS** will be the fastest but may give a "Step count" that is very high (not-complete).

3. **BFS** is reliable for finding the shortest path but is dangerous for large puzzles because it can quickly run out of RAM.

4. **A*** will give you the best balance of speed and a low "Step count."