8-Puzzle Solver Using A* Search

Project Description

Overview:
This project implements a solver for the 8-puzzle problem (and its generalizations to an n×n board) using the A* search algorithm. In the 8-puzzle, eight numbered tiles and one blank are arranged on a 3×3 grid. The goal is to slide the tiles—by moving a neighboring tile into the blank space—until the board is arranged in row-major order (with the blank in the last position). Note that not every board configuration is solvable. This implementation uses a *twin board* strategy (swapping two non-blank tiles) to detect unsolvability.

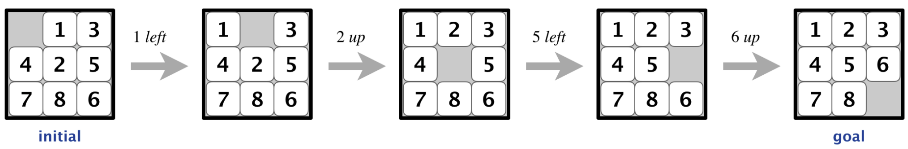

Figure 1: 8-Puzzle transitions from initial to goal

Key Components

1. Board Data Type

- **Representation:**
  The board is modeled as an immutable object containing an n×n grid. Tiles are stored where 0 represents the blank square.

Figure 2: Board representation and string representation

- **Methods:**

- **Constructor:** Initializes the board from an n×n array (or list of lists in Python).
- `toString()` / `__str__`**:** Returns a string representation that shows the board size and layout.
- **Heuristics:**
  * *Hamming distance:* Counts the number of tiles out of place.
  * *Manhattan distance:* Sums the vertical and horizontal distances from each tile to its goal position.

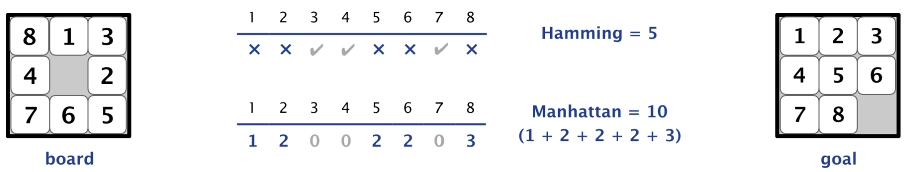

Figure 3: Hamming and Manhattan distances example

- `isGoal()`**:** Checks if the board matches the goal state.
- `neighbors()`**:** Generates all valid boards reachable by sliding a tile into the blank space.

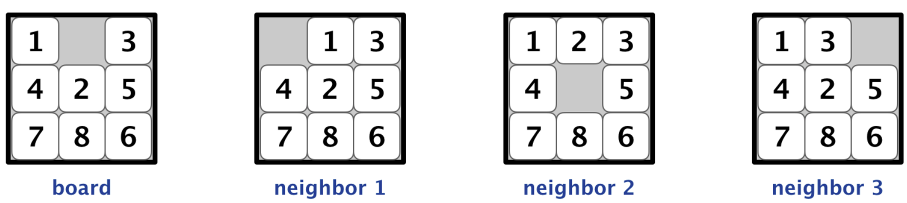

Figure 4: Board and neighbors

- `twin()`**:** Creates a board by swapping any pair of non-blank tiles (used to detect unsolvable puzzles).

2. Solver Data Type

- **Search Node:**
  Each search node stores:
  - A board.
  - The number of moves made so far.
  - A link to the previous node.
  - A priority computed as the sum of the moves and the board's Manhattan distance.

- **A* Search:**
  Uses a priority queue to always expand the node with the lowest priority. When the goal board is reached, the solution path is reconstructed.
- **Dual Search for Unsolvability:**
  Two searches are performed in lockstep—one on the initial board and one on its twin. If the twin reaches the goal, the original board is unsolvable.
- **Optimizations:**
  - Caching of the Manhattan (or Hamming) distances.
  - Pruning redundant search nodes by skipping neighbors that revert to the previous board.

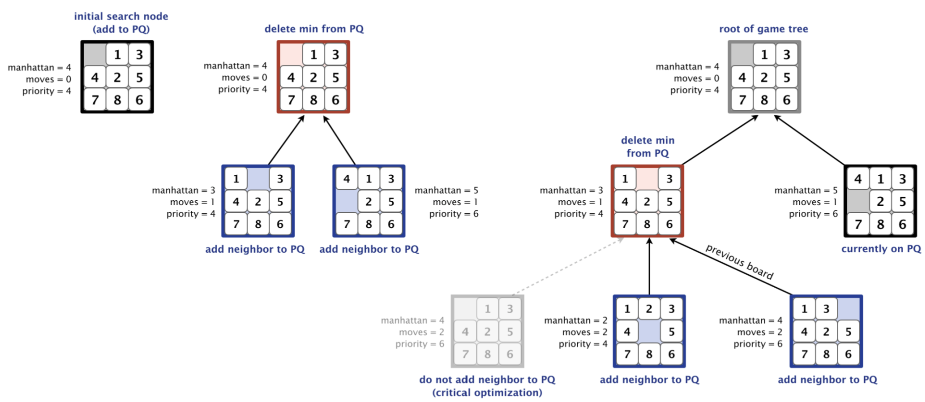

Figure 5: Search node expansions example

Hints on How to Solve the Problem

1. **Understanding the Board Representation:**
   - Model the board as an immutable data type so that once created, its state does not change. This helps in reliably comparing boards during the search.
   - Use a 2D array (or tuple of tuples) for representing the board configuration.
2. **Implementing Heuristics:**
   - **Hamming Distance:**
     Count how many tiles (ignoring the blank) are not in their goal position.
   - **Manhattan Distance:**
     For each tile, compute the distance from its current position to its target position (sum of horizontal and vertical differences).

- **Caching:**
  Since the Manhattan distance is computed repeatedly, calculate it once for each search node and store it.

3. **Generating Neighbors:**
   - Identify the position of the blank tile.
   - Create new board configurations by sliding a neighboring tile into the blank space (considering up, down, left, and right moves).
   - **Optimization:** Do not enqueue a neighbor if it is the same as the previous board (to avoid cycling back).
4. **A* Search Implementation:**
   - Use a priority queue (e.g., `heapq` in Python) to store search nodes.
   - Define each search node's priority as `moves + heuristic` (with the Manhattan heuristic preferred for better performance).
   - Expand the search by dequeuing the node with the smallest priority, then enqueue all its valid neighbors.
5. **Detecting Unsolvable Boards:**
   - Generate a twin board by swapping two non-blank tiles.
   - Run A* search simultaneously on both the initial board and its twin. If the twin board reaches the goal state, then the initial board is unsolvable.
6. **Reconstructing the Solution:**
   - When the goal state is reached in the search, backtrack using the linked search nodes to build the sequence of moves leading from the initial board to the goal.