State Struct (`State`): Represents the current state of the puzzle board, including the board configuration (`board`), position of the blank tile (`blank_row`, `blank_col`), and the current cost to reach this state (`cost`).

Priority Queue (`PriorityQueue`): Implemented to manage states based on their priority during the search process. Uses a simple array-based implementation for nodes (`PQNode`), where each node contains a pointer to a `State` and a priority value.

Functions:

initializeState: Initializes the initial state of the puzzle board from a given 2D array.

isGoalState: Checks if the current state matches the predefined goal state, where all tiles are in ascending order from 1 to 8 and the empty tile (`0`) is in the correct position.

Priority Calculation Functions:

calculateHammingPriority: Computes the Hamming priority (number of misplaced tiles) for a given state.

calculateManhattanDistance and calculateManhattanPriority: Computes the Manhattan distance (sum of distances of each tile from its goal position) and the Manhattan priority for a given state, respectively.

printBoard: Prints the current configuration of the puzzle board.

Utility Functions (swap, createPriorityQueue, push, pop, isEmpty): Manage operations on the priority queue such as inserting, removing, and checking for empty conditions.

3. Main Function (main):

Initialization: Defines the initial puzzle configuration and initializes the puzzle state using `initializeState`.

Goal State Check: Verifies if the initial state is already the goal state.

Print Initial State: Outputs the initial configuration of the puzzle board.

Priority Calculations: Computes and prints both Hamming and Manhattan priorities for the initial state.

A* Search: Executes the A* search algorithm to find the optimal solution path from the initial state to the goal state. It uses both Hamming and Manhattan priorities combined with the current cost to determine the priority of each state.

4. A* Search Function:

Initialization: Starts with the initial state pushed onto the priority queue.

Search Loop: Continues until the priority queue (`openSet`) is empty.

State Expansion: For each state popped from the queue, generates successor states by moving the blank tile in all valid directions (up, down, left, right).

Goal Check: Checks if the current state matches the goal state.

State Management: Allocates memory for each successor state, calculates its priority, updates the cost, and pushes it onto the priority queue.

Memory Management: Frees memory for each processed state after expansion.