Solving the N-Puzzle Using A* Search Algorithm

Github repository: **https://github.com/tmasmaliyev/N-Puzzle-GWU**

Introduction

The N-puzzle objective is to rearrange the numbered tiles in an $n \times n$ grid into the correct order using the fewest moves. The puzzle contains numbers from 1 to $n^2 - 1$ and a single blank space (represented as 0), which can be used to move adjacent tiles. This report describes the approach used to solve the N-puzzle using the A* search algorithm.

Problem Definition

Given an initial configuration of the N-puzzle in a text file, the program should find the shortest sequence of moves that leads to the solved configuration. The constraints for the puzzle are:

- The grid size $n\ x\ n$ ranges from 3 to 6.
- The input file contains n lines, each with n tab-separated numbers.
- The number 0 represents the blank space.
- Tiles can be moved into the blank space from adjacent positions (up, down, left, right).

Algorithm Choice and Approach

The A* search algorithm was chosen for its efficiency in finding the optimal solution by balancing exploration and path cost estimation. The approach involves:

1. Using a priority queue to prioritize states with the lowest cost estimate.
2. Applying the Manhattan distance heuristic to estimate the remaining cost to the goal.
3. Expanding states by moving tiles into the blank space and tracking visited states.
4. Constructing the shortest solution path using parent-child relationships.

This method ensures that the solution is found optimally while minimizing unnecessary computations.

Approach

The solution employs the A* search algorithm, which is a search method that uses both the cost to reach a state and an estimated cost to reach the goal. The key components of the algorithm are:

State Representation

Each state of the puzzle is represented as a 2D list containing the tile numbers. The blank tile position is tracked to determine possible moves.

Heuristic Function

The heuristic function guides the A* search to find an optimal solution efficiently. The chosen heuristic is the **Manhattan distance**, which calculates the sum of the horizontal and vertical distances each tile is from its goal position:

$$h(n) = \sum_{i=1}^{N} |x_i - x_g| + |y_i - y_g|$$

where (x_i, y_i) is the current position of tile i, and (x_g, y_g) is its goal position.

Search Algorithm

1. Initialize the priority queue with the initial state.
2. Expand the node with the lowest $f(n) = g(n) + h(n)$, where:
    - $g(n)$ is the cost to reach the current state.
    - $h(n)$ is the estimated cost to the goal.
3. Generate valid child states by moving tiles into the blank space.
4. Add new states to the priority queue if they have not been visited or have a lower cost.
5. Continue until the goal state is reached.

Implementation Details

- The input is read from a file such as:

  python main.py --filepath=./npuzzle/np5_27_7216.txt

- A priority queue (min-heap) is used to efficiently select the best node to expand.
- A set stores visited states to prevent re-exploration.
- The final path is reconstructed from the parent-child relationships stored during the search.

Results and Performance Analysis

The program was tested on various grid sizes $(3x3, 4x4, 5x5$. The execution time and number of moves required varied depending on the puzzle complexity. The A* search performed well due to the effective heuristic function.

Example Test Case

Input:

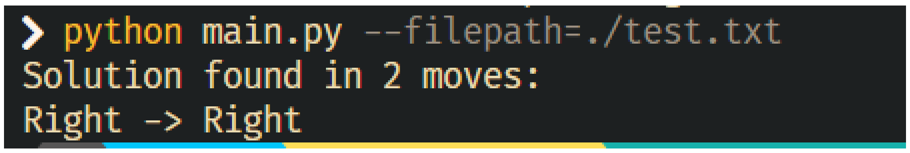
1	2	3
4	5	6
	7	8

Solution:

Moves required: 2

1. Move blank to right
2. Move blank to right

In the code example:



```
❯ python main.py --filepath=./test.txt
Solution found in 2 moves:
Right -> Right
```