Cory Mollenhour
CSCI 4350
Open Lab 1 - 8-Puzzle Problem
10/2/2018

8-Puzzle Problem

The 8-puzzle problem uses a hypothetical “board” with 8 total tiles and an empty slot. In order to solve the puzzle, the 8 square tiles must be slid across the board contained within walls, but not lifted, until the tiles align in the order specified. This goal is usually specified as *empty* - 1 - 2 - 3 - 4 - 5 - 6 - 7- 8. The board should always be solvable unless it was set up incorrectly.

The code I developed first generates a random board starting with a user specified configuration of the 9 spaces. From that configuration, the tiles are slid randomly (up, left, down, or right). With a higher seed, the moves are more randomized, and with a higher amount of moves, the solution will take longer to find. Then this board can be piped into the a-star program to find its solution. The a-star program takes in the board as input, and 2 command line arguments (seed #, and move #). The start board is then pushed onto the frontier and then the possible moves of the empty slot are pushed as well. Once the goal state matches the current board configuration of the node, the program then traces up the parent nodes to the original state to create the path.

The first heuristic (0) will always return zero and will not help guide the empty slot around the board to find the solution. However, a solution can be found without a useful heuristic, and the path will still be found by traversing the parent nodes to the original start position. The second heuristic (1) will implement an informed search, also known as the Hamming Function. It looks at the whole board at once and counts all tiles that are not in their correct position. This number is then compared to prioritize the lower count on the priority queue (the open list). The third heuristic is the Manhattan Function which prioritizes board configurations based on individual Manhattan distances of each tile on the board. These distances are totaled and provide a more intuitive cost for the board. The fourth heuristic was supposed to be a customized heuristic, but is not provided at this time.

When this code was first tested on my local computer, the informed search heuristic 1 took roughly 17 seconds to find a solution to a 100 move configuration. This was a large decrease in time from the original 315 seconds for a 30 move configuration. What made the greatest increase in speed was by utilizing an unordered map for the closed list, which is compared with each node. If a vector is used to iterate through, this would become exponentially longer with each node depth reached. If the empty space is included in the board configuration comparisons, this also increases the time to find a solution by a significant amount. Also, avoiding nested for loops for comparing board configurations increased speed significantly as this would be done many times.To avoid that issue, the 2D arrays were also implemented as 1D strings. Overall the Manhattan function performed significantly faster than the other two, as the heuristic was more designed for the equal distance square setup of the board.

Cory Mollenhour
CSCI 4350
Open Lab 1 - 8-Puzzle Problem
10/2/2018

Using a seed of 100 and 100 moves the following statistics were gathered:
h0 -

- Nodes Visited (V) - 143,695
- Nodes In Memory (N) - 139,118
- Depth of Solution (d) - 22
- Branch Factor (b) - 1.713129

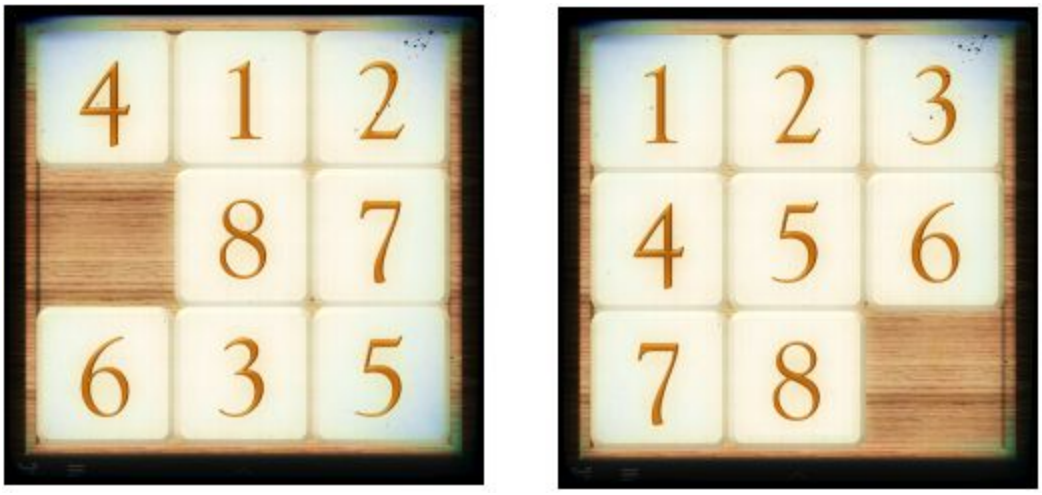
h1 -

- Nodes Visited (V) - 4,823
- Nodes In Memory (N) - 7,432
- Depth of Solution (d) - 22
- Branch Factor (b) - 1.499545

h2 -

- Nodes Visited (V) - 164
- Nodes In Memory (N) - 263
- Depth of Solution (d) - 22
- Branch Factor (b) - 1.288244

h3 - Unavailable



8-Puzzle Board (image credit: http://remondo.net/)