Project 1: 8-Puzzle Solver

Talor Anderson - CS 4200

The 8-puzzle problem is an interesting problem for artificial intelligence. It can be solved using the A* algorithm.

Implementation Overview

My project is divided into a few key files. Here is a brief description of each.

Tester Class

This class holds the `main` Java method and handles the menu system, user input, and all file I/O.

Generator

This class holds all static methods for generating puzzles.

Solver

This class holds the key `solve()` method and does goal testing. Both are implemented statically.

The solve method is implemented with a Java `HashSet<Node>` of as the explored set and a Java `PriorityQueue<Node>` as the frontier.

Node

Holds data for one puzzle state. Includes methods for calculating f, g, h1 and h2, and n.

Solution

An instance of this class is returned by the Solver.solve() method. It holds both the final solved puzzle node as well as fields for execution time, number of generated nodes, and solution depth.

Results

These figures were created by appending 1000 rows to a .csv file (menu option 4) of completely randomly generated solved puzzles. Then, I analyzed the outputted data to get the following statistics. The `Generator` class created puzzles with depths ranging from 10-30. Execution was very slow given that the solution only runs on a single thread, and a better implementation would solve many puzzles in parallel. Because of the long execution time these figures only represent heuristic 1, though both are implemented.

```
depth    cost         time(ms)

10         80               0
12        229               0
14        586               0
16       1357               2
18       3189              13
20       8396              92
22      20808             580
24      46638            2804
26      99934           12326
28     165882           34411
30     255591           82240
```

Sample Output

```
Enter a Selection:
1) Solve a random puzzle
2) Input a puzzle to solve
3) Solve 1000 puzzles from puzzles.txt
4) run cases and output to CSV
5) quit

-> 2

Enter puzzle:
-> 1 6 2 8 0 4 3 7 5

[ 1 6 2 8 0 4 3 7 5 ]
[ 1 6 2 0 8 4 3 7 5 ]
[ 1 6 2 3 8 4 0 7 5 ]
[ 1 6 2 3 8 4 7 0 5 ]
[ 1 6 2 3 0 4 7 8 5 ]
```

```
[ 1 0 2 3 6 4 7 8 5 ]
[ 0 1 2 3 6 4 7 8 5 ]
[ 3 1 2 0 6 4 7 8 5 ]
[ 3 1 2 6 0 4 7 8 5 ]
[ 3 1 2 6 4 0 7 8 5 ]
[ 3 1 2 6 4 5 7 8 0 ]
[ 3 1 2 6 4 5 7 0 8 ]
[ 3 1 2 6 4 5 0 7 8 ]
[ 3 1 2 0 4 5 6 7 8 ]
[ 0 1 2 3 4 5 6 7 8 ]
depth:   14, cost:    704 nodes, time: 11ms

Enter puzzle:
-> 0 4 5 2 1 8 3 6 7

[ 0 4 5 2 1 8 3 6 7 ]
[ 2 4 5 0 1 8 3 6 7 ]
[ 2 4 5 1 0 8 3 6 7 ]
[ 2 0 5 1 4 8 3 6 7 ]
[ 0 2 5 1 4 8 3 6 7 ]
[ 1 2 5 0 4 8 3 6 7 ]
[ 1 2 5 3 4 8 0 6 7 ]
[ 1 2 5 3 4 8 6 0 7 ]
[ 1 2 5 3 4 8 6 7 0 ]
[ 1 2 5 3 4 0 6 7 8 ]
[ 1 2 0 3 4 5 6 7 8 ]
[ 1 0 2 3 4 5 6 7 8 ]
[ 0 1 2 3 4 5 6 7 8 ]
depth:   12, cost:    237 nodes, time: 2ms

Enter puzzle:
-> 6 5 0 1 3 4 7 2 8

[ 6 5 0 1 3 4 7 2 8 ]
[ 6 0 5 1 3 4 7 2 8 ]
[ 6 3 5 1 0 4 7 2 8 ]
[ 6 3 5 0 1 4 7 2 8 ]
[ 0 3 5 6 1 4 7 2 8 ]
[ 3 0 5 6 1 4 7 2 8 ]
[ 3 1 5 6 0 4 7 2 8 ]
[ 3 1 5 6 2 4 7 0 8 ]
[ 3 1 5 6 2 4 0 7 8 ]
[ 3 1 5 0 2 4 6 7 8 ]
[ 0 1 5 3 2 4 6 7 8 ]
[ 1 0 5 3 2 4 6 7 8 ]
[ 1 2 5 3 0 4 6 7 8 ]
```

```
[ 1 2 5 3 4 0 6 7 8 ]
[ 1 2 0 3 4 5 6 7 8 ]
[ 1 0 2 3 4 5 6 7 8 ]
[ 0 1 2 3 4 5 6 7 8 ]
depth:   16, cost:   1229 nodes, time: 10ms
```