

Jianbang Chen

Levi Santana de Lelis

CMPUT 366

2021.12.03

Implement IDA* for solving the 3x3 Sliding-Tile puzzle

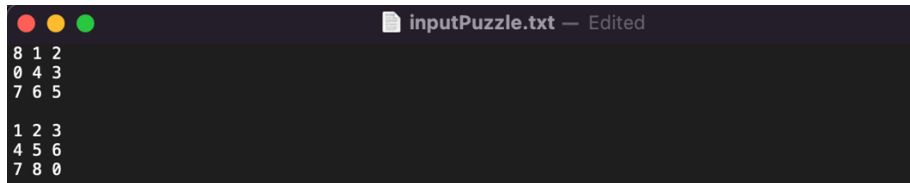
Introduction: This project is aimed to solve the 3x3 Sliding-Tile puzzle by implement Iterative-Deepening A* (IDA*). The game has a small platform with generally 9 slots in a 3 by 3 square. There are 8 tiles usually numbered from 1 to 8, one per slot. This leaves an empty slot. Any tile adjacent to the empty slot may slide into it, which then makes the tiles previous slot empty. The object is to slide tiles, one at a time, to bring them to goal state. Iterative deepening A* (IDA*) is a graph traversal and path search algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph. IDA* uses the more informative $f(n) = g(n) + h(n)$. The IDA* is important when we try to solve large size Sliding-Tile puzzle, IDA* can perform much better than other search algorithms such as Dijkstra's algorithm and BFS.

Methods: For 3x3 Sliding-Tile puzzle problem I use the Iterative-Deepening A*(IDA*) algorithm with sum of Manhattan Distance of all tiles to derive a heuristic function. Manhattan Distance is the distance between two points measured along axes at right angles. In a plane with p1 at (x1, y1) and p2 at (x2, y2), it is $|x1 - x2| + |y1 - y2|$. In addition, since some Sliding-Tile puzzle problem may not solvable we also need to check number of inversions for the puzzle. If number of inversions is odd in the initial state, then this puzzle is not solvable. Inversion is A pair of tiles form an inversion if the values on tiles are in reverse order of their appearance in goal state. The inputPuzzle.txt will store the initial state and goal state for the 3x3 Sliding-Tile puzzle (number 0 represent empty). The puzzle on the top is the initial state and the puzzle on the bottom is the goal state. When main.py runned, it will read from the input file and store the puzzle to initial state puzzle and goal state puzzle. Then it will check if the puzzle is solvable by counting the inversions of the initial puzzle. If the puzzle is not solvable then it will directly

return fail message. If the puzzle is solvable then it will apply IDA* with sum of Manhattan Distance of all tiles to solve the puzzle. The solving process will print out with a solve success message.

Evaluation: In the terminal run with command “python3 main.py”

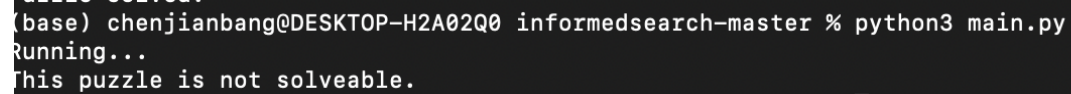
When the puzzle is not solvable since the inversion is 11 an odd number:



```
inputPuzzle.txt — Edited
8 1 2
0 4 3
7 6 5

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

Output:



```
(base) chenjianbang@DESKTOP-H2A02Q0 informedsearch-master % python3 main.py
Running...
This puzzle is not solveable.
```

When puzzle is solvable:

```
inputPuzzle.txt
1 2 3
4 0 5
8 6 7

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

Output:

```
(base) chenjianbang@DESKTOP-H2A02Q0 informedsearch-master % python3 main.py
Running...
1 2 3
4 0 5
8 6 7

1 2 3
4 5 0
8 6 7

1 2 3
4 5 7
8 6 0

1 2 3
4 5 7
8 0 6

1 2 3
4 5 7
0 8 6

1 2 3
0 5 7
4 8 6

1 2 3
5 0 7
4 8 6

1 2 3
5 7 0
4 8 6

1 2 3
5 7 6
4 8 0

1 2 3
5 7 6
4 0 8

1 2 3
5 0 6
4 7 8

1 2 3
0 5 6
4 7 8

1 2 3
4 5 6
0 7 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

Puzzle solved.
```

Works Cited

Ishan. "How to Check If an Instance of 8 Puzzle Is Solvable?" GeeksforGeeks, Ishan, 3 Nov. 2021,

<https://www.geeksforgeeks.org/check-instance-8-puzzle-solvable/>.

Paul E. Black, "Manhattan distance", in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed. 11 February 2019. (accessed TODAY) Available from: <https://www.nist.gov/dads/HTML/manhattanDistance.html>