15 Puzzle Solver

Kavana Anand, James Cingone, Tanvi Sirsat

The Problem

The 15-puzzle is a sliding tile puzzle, which consists of a 4x4 matrix of 15 tiles numbered 1 to 15 in an unordered manner and one blank space. The objective of the puzzle is to arrange the tiles in numeric order by moving them one by one, utilizing the blank space to change between states. It is a variant of the n-puzzle, a famous problem in computer science incorporating heuristics.
Previous studies have concluded that puzzles containing an odd number of permutations are unsolvable, while all puzzles containing even permutations are always solvable (Johnson and Story 1879.) Therefore, after one has identified the nature of the puzzle's permutation, returning an appropriate answer is straightforward.

Plan

Our aim is to design an algorithm that takes a randomized puzzle as an initial state, evaluates its solvability and calculates an optimal path to reach the goal, which is a solved puzzle pattern. We will start by researching practical algorithms for completing a randomized puzzle and quickly graduate to more sophisticated, broad formulae. Implementing these concepts through Prolog should prove to be a challenging task; however, once we have programmed a working algorithm we will immediately move to optimization so we may complete the puzzle in as few steps as possible.

Our primary action in this problem would be 'move', which specifies the movement of the tile. To be more specific, our actions will be the movement of a tile in a specific direction.

Our plan is to input an initial state and output a list of movements of the tiles relative to the empty space in order to achieve the goal [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0]. Here, 0 represents the blank space.

So if our query is of the form -

-
  ```
  ?- answer([1 2 3 4 5 6 7 8 9 10 11 12 13 0 14 15], L).
  output : L = [right right];
  ```

-
  ```
  ?- answer([1 2 3 4 5 6 7 8 9 10 12 15 13 0 14 11], L).
  output : L = [right right up left down right];
  ```

-
  ```
  ?- answer([1 3 2 4 8 6 7 5 12 10 11 9 13 14 15], L).
  ```

```
output : false;
```

The last query should return false, as the pattern in the initial state is unsolvable.

We intend for the program to be capable of recursively solving for multiple solutions sorted by length. However, our goals regarding multiple solutions may change as we broaden our base of knowledge.

Milestones / Development plan

- Locate the tile 1 (tile to be placed in first position) and place it in its correct position

  Finding the blank space and moving tiles to get blank space closer to tile 1 achieve this.

- Placing all tiles in a row in their correct positions

  In 15 puzzle, all the 4 tiles in first row will be placed in their positions by following the above technique for tile 2 and getting blank space to fourth position and tile 4 to third position and tile 3 under its actual position. We do this for the next row as well.

- Placing all tiles in first column in their correct positions

  In 15 puzzle, all the 4 tiles in first column will be placed in their positions. Since first two tiles of the column is already placed. We move the blank space among the last two rows and place the last two tiles of the column in their respective position by using the similar technique which we used to place the tiles 3 and 4 above. Same for the last two tiles in next column.

- Solution to the problem

  This includes giving the final output to the problem that gives us the movement of the blank tile to achieve the goal state, a solved puzzle pattern.

- Optimization of number of steps involved in moving a tile to its correct position

  This involves finding the minimum of number of moves required to get a tile from its current position to its original position among many different solutions to it.

- Check for solvability

  Will return false if goal state can't be achieved from the given initial state.

Test cases / Evaluation of the project success

Our test cases mostly follow what is achieved in the milestones -

- Given an initial state, test for the solution of the first tile to be placed in the first position from its current position.
- Given an initial state, test for the solution of the first row which also tests the second as well as both follow the same technique.
- Given an initial state with first two rows i.e, first eight tiles in their correct positions, test for the solution of the placing the last two tiles(tile 9 and tile 13) of the first column. This will also test the placing of the tile 10 and 14.
- Final test case - Given an initial state, test for the solution of the problem which will return the movement of the blank tile in order to achieve the goal.
- Solvability test - Given an initial state that is unsolvable, the program should return false.

Project Progress

The final code begins by calculating the shortest possible path to the goal state with respect to the blank space, utilizing the Manhattan Distance heuristic. After calculating an optimal solution, the program prints a graphical representation of the start, goal and intermediate states, and presents its solution; a list of movements of the blank space. Multiple, sub-optimal solutions can be calculated in this way.

In the following paragraphs, we present a number of test solutions -

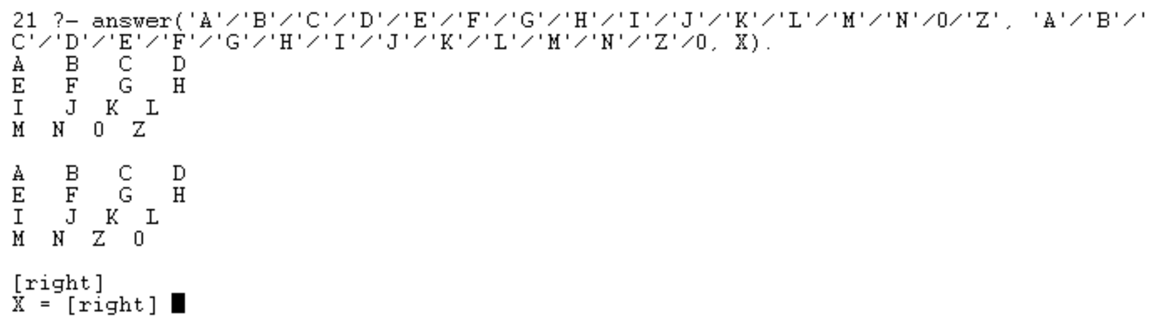
Test 1: Fundamental Movement

Our first test presents the program's basic capability to detect disparity between the start and goal states taken as arguments, as well as the functionality of its movement heuristics.

Start State: A/B/C/D/E/F/G/H/I/J/K/L/M/N/0/Z

Goal State: A/B/C/D/E/F/G/H/I/J/K/L/M/N/Z/0

Notes: This is a 1-move solution that moves tile 15 one space to its final destination. Each letter corresponds to its lexicographical order, with the exception of Z (15) to prevent any confusion between 0 and O.



```
21 ?- answer('A'/'B'/'C'/'D'/'E'/'F'/'G'/'H'/'I'/'J'/'K'/'L'/'M'/'N'/0/'Z', 'A'/'B'/'
C'/'D'/'E'/'F'/'G'/'H'/'I'/'J'/'K'/'L'/'M'/'N'/'Z'/0, X).
A   B   C   D
E   F   G   H
I   J  K  L
M  N  0  Z

A   B   C   D
E   F   G   H
I   J  K  L
M  N  Z  0

[right]
X = [right] █
```

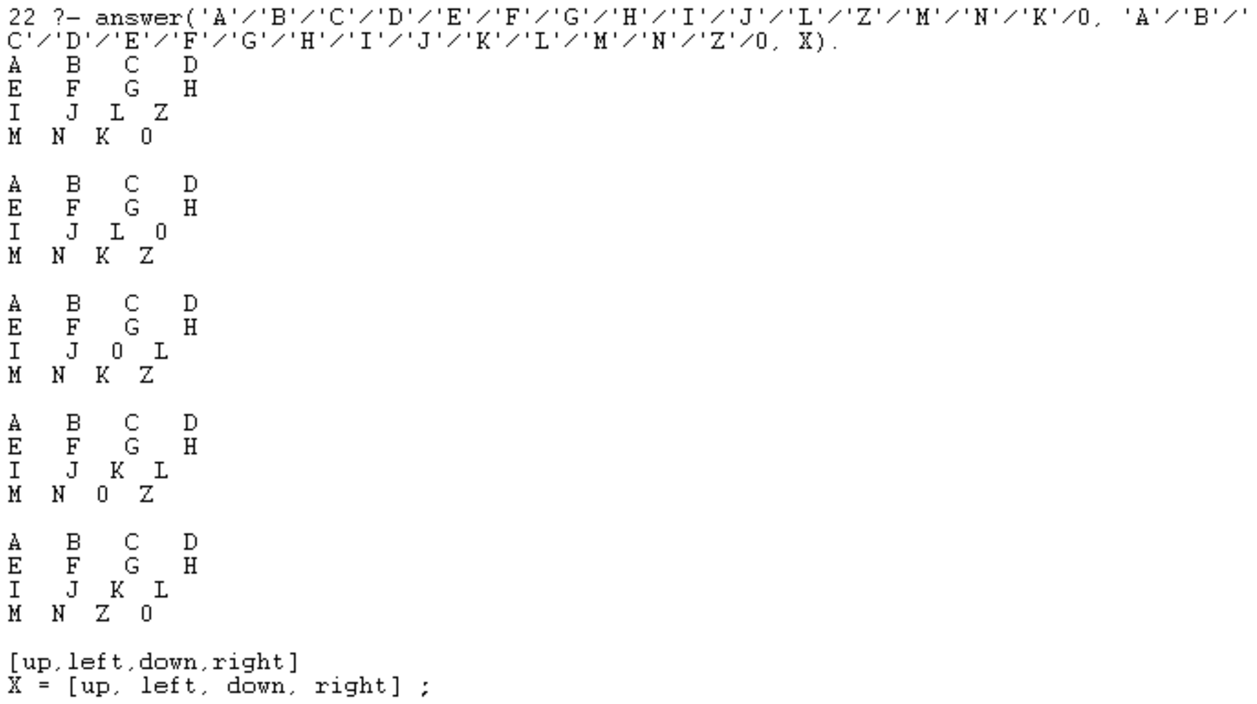
Result: *Success.*

Test 2: Tile Interaction and Optimization

Start State: A/B/C/D/E/F/G/H/I/J/L/Z/M/N/K/0

Goal State: A/B/C/D/E/F/G/H/I/J/K/L/M/N/Z/0

Notes: This is a 4-move solution meant to test not only the program's ability to detect a solve involving multiple goal states, but to determine its ability to detect the most optimal solution possible. The program must go through various fail states before reaching a solution, and must recurse appropriately.



```
22 ?- answer('A'/'B'/'C'/'D'/'E'/'F'/'G'/'H'/'I'/'J'/'L'/'Z'/'M'/'N'/'K'/0, 'A'/'B'/'
C'/'D'/'E'/'F'/'G'/'H'/'I'/'J'/'K'/'L'/'M'/'N'/'Z'/0, X).
A   B   C   D
E   F   G   H
I   J  L  Z
M  N  K  0

A   B   C   D
E   F   G   H
I   J  L  0
M  N  K  Z

A   B   C   D
E   F   G   H
I   J  0  L
M  N  K  Z

A   B   C   D
E   F   G   H
I   J  K  L
M  N  0  Z

A   B   C   D
E   F   G   H
I   J  K  L
M  N  Z  0

[up,left,down,right]
X = [up, left, down, right] ;
```

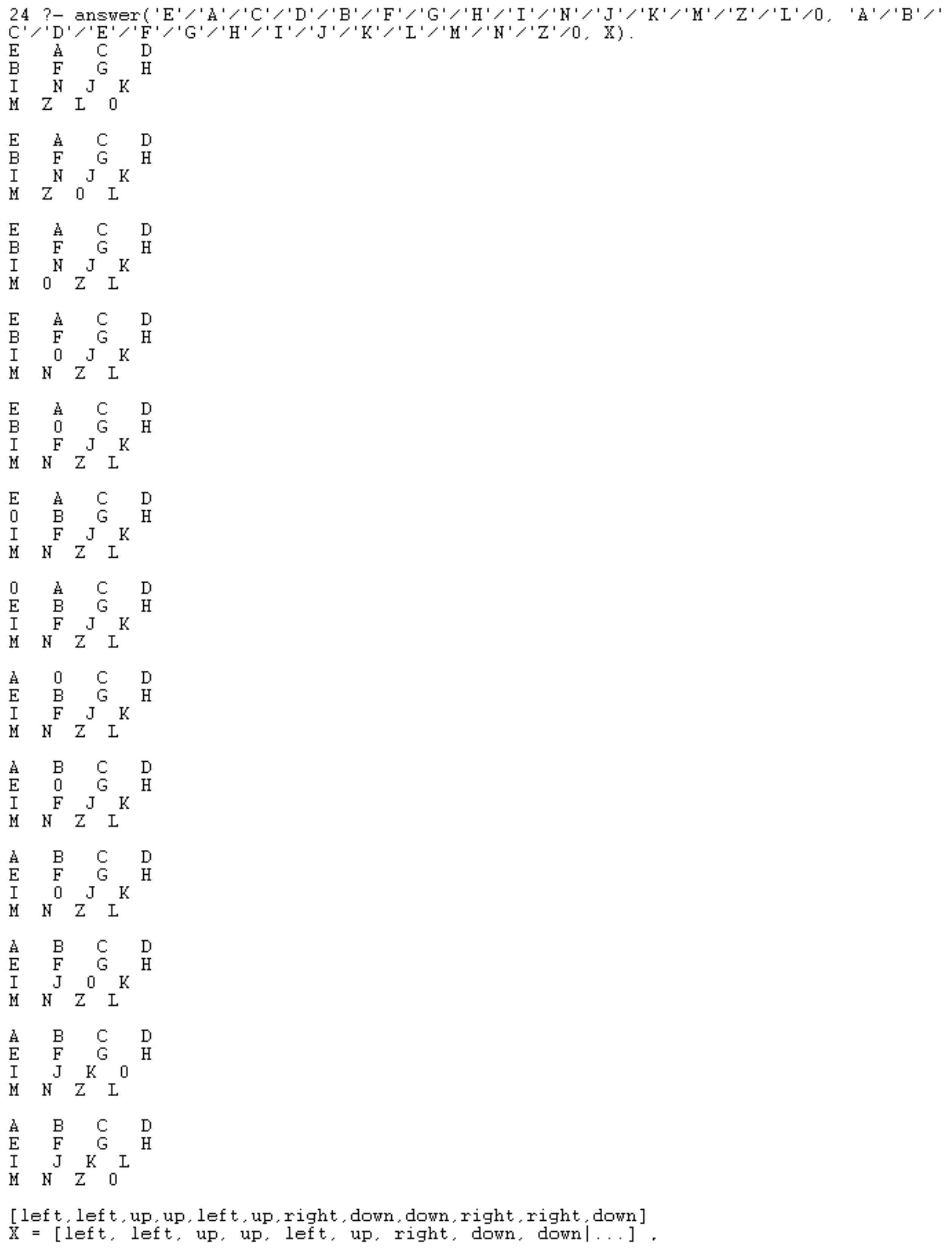
Result: *Success.*

Test 3*: *Complex Randomized Puzzles

Start State: E/A/C/D/B/F/G/H/I/N/J/K/M/Z/L/0

Goal State: A/B/C/D/E/F/G/H/I/J/K/L/M/N/Z/0

Notes: This is a fully randomized puzzle kept down to a 10-step solution, meant to test the puzzle's solving ability in a practical situation.



```
24 ?- answer('E'/'A'/'C'/'D'/'B'/'F'/'G'/'H'/'I'/'N'/'J'/'K'/'M'/'Z'/'L'/0, 'A'/'B'/'
C'/'D'/'E'/'F'/'G'/'H'/'I'/'J'/'K'/'L'/'M'/'N'/'Z'/0, X).
E   A   C   D
B   F   G   H
I   N  J  K
M  Z  L  0

E   A   C   D
B   F   G   H
I   N  J  K
M  Z  0  L

E   A   C   D
B   F   G   H
I   N  J  K
M  0  Z  L

E   A   C   D
B   F   G   H
I   0  J  K
M  N  Z  L

E   A   C   D
B   0   G   H
I   F  J  K
M  N  Z  L

E   A   C   D
0   B   G   H
I   F  J  K
M  N  Z  L

0   A   C   D
E   B   G   H
I   F  J  K
M  N  Z  L

A   0   C   D
E   B   G   H
I   F  J  K
M  N  Z  L

A   B   C   D
E   0   G   H
I   F  J  K
M  N  Z  L

A   B   C   D
E   F   G   H
I   0  J  K
M  N  Z  L

A   B   C   D
E   F   G   H
I   J  0  K
M  N  Z  L

A   B   C   D
E   F   G   H
I   J  K  0
M  N  Z  L

A   B   C   D
E   F   G   H
I   J  K  L
M  N  Z  0

[left,left,up,up,left,up,right,down,down,right,right,down]
X = [left, left, up, up, left, up, right, down, down|...] .
```

Result: Success.