Delhi Technological University



Department of Computer Science and Engineering

B. Tech (Computer Engineering)

VI Semester 2020-21

Artificial Intelligence CO302

Assignment 2

SUBMITTED BY:

Prithish Kumar Rath

2K18/CO/262

1. **Write down two intelligent programs for the TIC-TAC-TOE problem.**
2. **Minmax Algorithm**

The classic Tic-Tac-Toe game (also called Noughts and Crosses) or Xs and Os is a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner.

In this code, I've used minimax algorithm to help the computer where to go for the next move and win the puzzle

**Output:**

Choose X or O

Chosen: X

First to start?[y/n]: y

Human turn [X]

---------------

| || || |

---------------

| || || |

---------------

| || || |

---------------

Use numpad (1..9): 4

Computer turn [O]

---------------

| || || |

---------------

| X || || |

---------------

| || || |

---------------

Human turn [X]

---------------

| O || || |

---------------

| X || || |

---------------

| || || |

---------------

Use numpad (1..9): 6

Computer turn [O]

---------------

| O || || |

---------------

| X || || X |

---------------

| || || |

---------------

Human turn [X]

---------------

| O || || |

---------------

| X || O || X |

---------------

| || || |

---------------

Use numpad (1..9): 9

Computer turn [O]

---------------

| O || || |

---------------

| X || O || X |

---------------

| || || X |

---------------

Human turn [X]

---------------

| O || || O |

---------------

| X || O || X |

---------------

| || || X |

---------------

Use numpad (1..9): 2

Computer turn [O]

---------------

| O || X || O |

---------------

| X || O || X |

---------------

| || || X |

---------------

Computer turn [O]

---------------

| O || X || O |

---------------

| X || O || X |

---------------

| O || || X |

---------------

YOU LOSE!

### **Combinatorics :**

When considering only the state of the board, and after taking into account board symmetries (i.e. rotations and reflections), there are only 138 terminal board positions. A combinatorics study of the game shows that when "X" makes the first move every time, the game is won as follows :

91 distinct positions are won by (X)

44 distinct positions are won by (O)

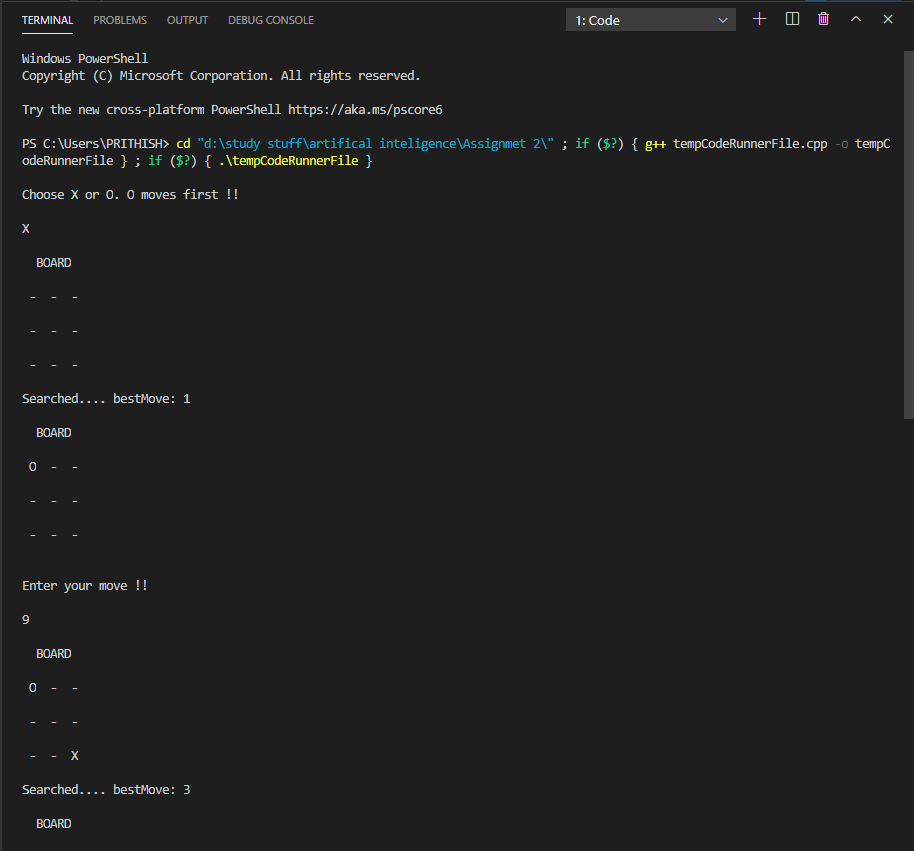
3 distinct positions are drawn (often called a "cat's game")

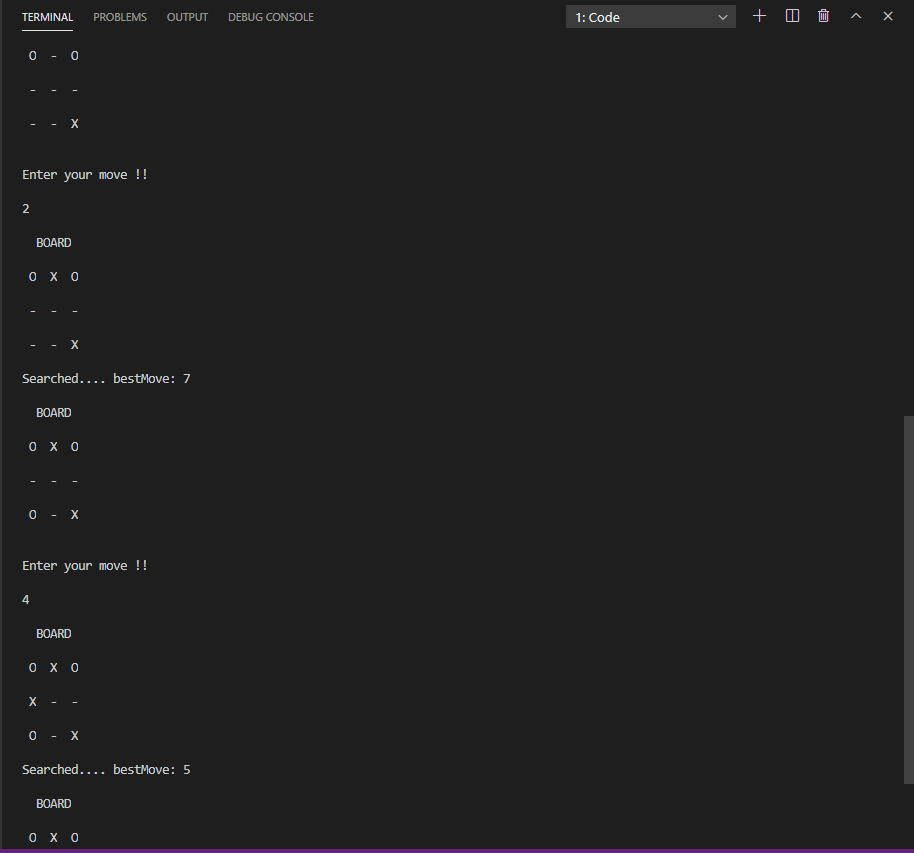
A C-implementation solving the 8-puzzle problem using the uninformed search strategy BFS (Breadth-First Search) and heusitic search strategy A\*. The goal is to empirically compare both strategies' space and time performance.

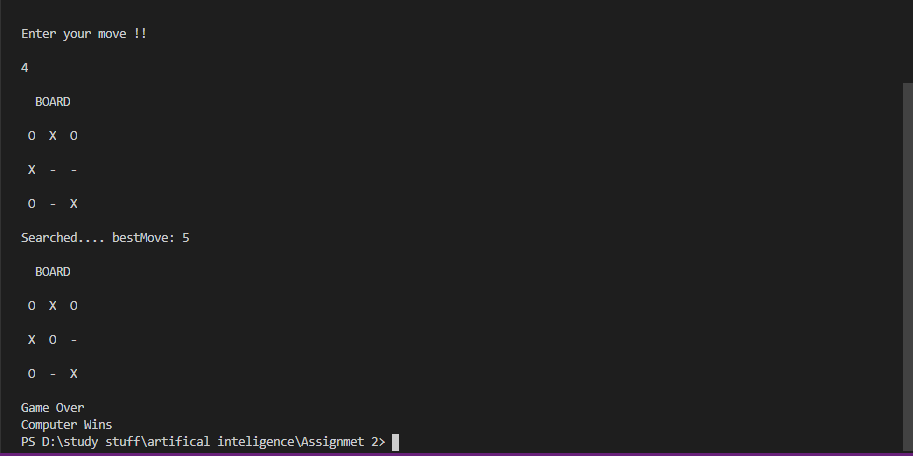
For each strategy, the program collects and outputs the following information:

* sequence of moves corresponding to the solution (e.g. up, down, left, right)
* total number of nodes expanded
* total number of nodes generated
* length of the solution path (number of moves)

**Output:**







1. **Reinforcement learning**

**Reinforcement learning** is a **Machine Learning** paradigm oriented on agents learning to take the best decisions in order to maximize a reward.

**Output:**

training...

Rounds 0

Rounds 1000

Rounds 2000

Rounds 3000

Rounds 4000

Rounds 5000

Rounds 6000

Rounds 7000

Rounds 8000

Rounds 9000

Rounds 10000

Rounds 11000

Rounds 12000

Rounds 13000

Rounds 14000

Rounds 15000

Rounds 16000

Rounds 17000

Rounds 18000

Rounds 19000

Rounds 20000

Rounds 21000

Rounds 22000

Rounds 23000

Rounds 24000

Rounds 25000

Rounds 26000

Rounds 27000

Rounds 28000

Rounds 29000

Rounds 30000

Rounds 31000

Rounds 32000

Rounds 33000

Rounds 34000

Rounds 35000

Rounds 36000

Rounds 37000

Rounds 38000

Rounds 39000

Rounds 40000

Rounds 41000

Rounds 42000

Rounds 43000

Rounds 44000

Rounds 45000

Rounds 46000

Rounds 47000

Rounds 48000

Rounds 49000

-------------

| x | | |

-------------

| | | |

-------------

| | | |

-------------

Input your action row:0

Input your action col:1

-------------

| x | o | |

-------------

| | | |

-------------

| | | |

-------------

-------------

| x | o | |

-------------

| | x | |

-------------

| | | |

-------------

Input your action row:2

Input your action col:2

-------------

| x | o | |

-------------

| | x | |

-------------

| | | o |

-------------

-------------

| x | o | |

-------------

| x | x | |

-------------

| | | o |

-------------

Input your action row:1

Input your action col:2

-------------

| x | o | |

-------------

| x | x | o |

-------------

| | | o |

-------------

-------------

| x | o | |

-------------

| x | x | o |

-------------

| x | | o |

-------------

computer wins!

1. **Write down a program to implement Breadth first search and depth first search for water jug problem.**

You are given a m litre jug and a n litre jug . Both the jugs are initially empty. The jugs don’t have markings to allow measuring smaller quantities. You have to use the jugs to measure d litres of water where d is less than n. (X, Y) corresponds to a state where X refers to amount of water in Jug1 and Y refers to amount of water in Jug2.

We basically perform three operations to achieve the goal.

Fill water jug.

Empty water jug

and Transfer water jug

We run breadth first search on the states and these states will be created after applying allowed operations and we also use visited map of pair to keep track of states that should be visited only once in the search. This solution can also be achieved using depth first search.

Depth -First Search algorithm is a graph traversing algorithm, where you select a random initial node (source or root node) and starts traversing the graph from root node and then goes to deeper and deeper until we find the goal node or the node which has no children.

**Output for Breadth first search:**

Receiving the volume of the jugs...

Enter first jug volume (>1): 4

Enter second jug volume (>1): 3

Receiving the desired amount of the water...

Enter the desired amount of water (1 - 4): 2

Enter 'b' for BFS, 'd' for DFS: b

Implementing BFS...

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 0] is visited before...

Checking if [0, 3] is visited before...

Checking if [0, 0] is visited before...

Checking if [0, 0] is visited before...

Checking if [0, 0] is visited before...

Checking if [0, 0] is visited before...

Possible transitions:

[4, 0]

[0, 3]

Fill 4-liter jug: [4, 0]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 0] is visited before...

Checking if [4, 3] is visited before...

Checking if [4, 0] is visited before...

Checking if [1, 3] is visited before...

Checking if [0, 0] is visited before...

Checking if [4, 0] is visited before...

Possible transitions:

[4, 3]

[1, 3]

Fill 3-liter jug: [0, 3]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 3] is visited before...

Checking if [0, 3] is visited before...

Checking if [3, 0] is visited before...

Checking if [0, 3] is visited before...

Checking if [0, 3] is visited before...

Checking if [0, 0] is visited before...

Possible transitions:

[4, 3]

[3, 0]

Fill 3-liter jug: [4, 3]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 3] is visited before...

Checking if [4, 3] is visited before...

Checking if [4, 3] is visited before...

Checking if [4, 3] is visited before...

Checking if [0, 3] is visited before...

Checking if [4, 0] is visited before...

No more unvisited nodes...

Backtracking...

Pour 4-liter jug into 3-liter jug: [1, 3]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 3] is visited before...

Checking if [1, 3] is visited before...

Checking if [4, 0] is visited before...

Checking if [1, 3] is visited before...

Checking if [0, 3] is visited before...

Checking if [1, 0] is visited before...

Possible transitions:

[1, 0]

Fill 4-liter jug: [4, 3]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 3] is visited before...

Checking if [4, 3] is visited before...

Checking if [4, 3] is visited before...

Checking if [4, 3] is visited before...

Checking if [0, 3] is visited before...

Checking if [4, 0] is visited before...

No more unvisited nodes...

Backtracking...

Pour 3-liter jug into 4-liter jug: [3, 0]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 0] is visited before...

Checking if [3, 3] is visited before...

Checking if [3, 0] is visited before...

Checking if [0, 3] is visited before...

Checking if [0, 0] is visited before...

Checking if [3, 0] is visited before...

Possible transitions:

[3, 3]

Clear 3-liter jug: [1, 0]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 0] is visited before...

Checking if [1, 3] is visited before...

Checking if [1, 0] is visited before...

Checking if [0, 1] is visited before...

Checking if [0, 0] is visited before...

Checking if [1, 0] is visited before...

Possible transitions:

[0, 1]

Fill 3-liter jug: [3, 3]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 3] is visited before...

Checking if [3, 3] is visited before...

Checking if [4, 2] is visited before...

Checking if [3, 3] is visited before...

Checking if [0, 3] is visited before...

Checking if [3, 0] is visited before...

Possible transitions:

[4, 2]

Pour 4-liter jug into 3-liter jug: [0, 1]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 1] is visited before...

Checking if [0, 3] is visited before...

Checking if [1, 0] is visited before...

Checking if [0, 1] is visited before...

Checking if [0, 1] is visited before...

Checking if [0, 0] is visited before...

Possible transitions:

[4, 1]

Pour 3-liter jug into 4-liter jug: [4, 2]

Checking if the gaol is achieved...

The goal is achieved

Printing the sequence of the moves...

Starting from: [0, 0]

1 : Fill 3-liter jug: [0, 3]

2 : Pour 3-liter jug into 4-liter jug: [3, 0]

3 : Fill 3-liter jug: [3, 3]

4 : Pour 3-liter jug into 4-liter jug: [4, 2]

**Output for Depth first search:**

Receiving the volume of the jugs...

Enter first jug volume (>1): 4

Enter second jug volume (>1): 3

Receiving the desired amount of the water...

Enter the desired amount of water (1 - 4): 2

Enter 'b' for BFS, 'd' for DFS: d

Implementing DFS...

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 0] is visited before...

Checking if [0, 3] is visited before...

Checking if [0, 0] is visited before...

Checking if [0, 0] is visited before...

Checking if [0, 0] is visited before...

Checking if [0, 0] is visited before...

Possible transitions:

[4, 0]

[0, 3]

Fill 3-liter jug: [0, 3]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 3] is visited before...

Checking if [0, 3] is visited before...

Checking if [3, 0] is visited before...

Checking if [0, 3] is visited before...

Checking if [0, 3] is visited before...

Checking if [0, 0] is visited before...

Possible transitions:

[4, 3]

[3, 0]

Pour 3-liter jug into 4-liter jug: [3, 0]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 0] is visited before...

Checking if [3, 3] is visited before...

Checking if [3, 0] is visited before...

Checking if [0, 3] is visited before...

Checking if [0, 0] is visited before...

Checking if [3, 0] is visited before...

Possible transitions:

[4, 0]

[3, 3]

Fill 3-liter jug: [3, 3]

Checking if the gaol is achieved...

Finding next transitions and checking for the loops...

Checking if [4, 3] is visited before...

Checking if [3, 3] is visited before...

Checking if [4, 2] is visited before...

Checking if [3, 3] is visited before...

Checking if [0, 3] is visited before...

Checking if [3, 0] is visited before...

Possible transitions:

[4, 3]

[4, 2]

Pour 3-liter jug into 4-liter jug: [4, 2]

Checking if the gaol is achieved...

The goal is achieved

Printing the sequence of the moves...

Starting from: [0, 0]

1 : Fill 3-liter jug: [0, 3]

2 : Pour 3-liter jug into 4-liter jug: [3, 0]

3 : Fill 3-liter jug: [3, 3]

1. : Pour 3-liter jug into 4-liter jug: [4, 2]
2. **Implement Best first search and least cost search for 8-Puzzle problem**

**Output of best first search**

RBFS: ['U', 'R', 'U', 'L', 'D']

space: 16

time: 0.0009970664978027344

------------------------------------------

RBFS: ['U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']

space: 29

time: 0.00099945068359375

------------------------------------------

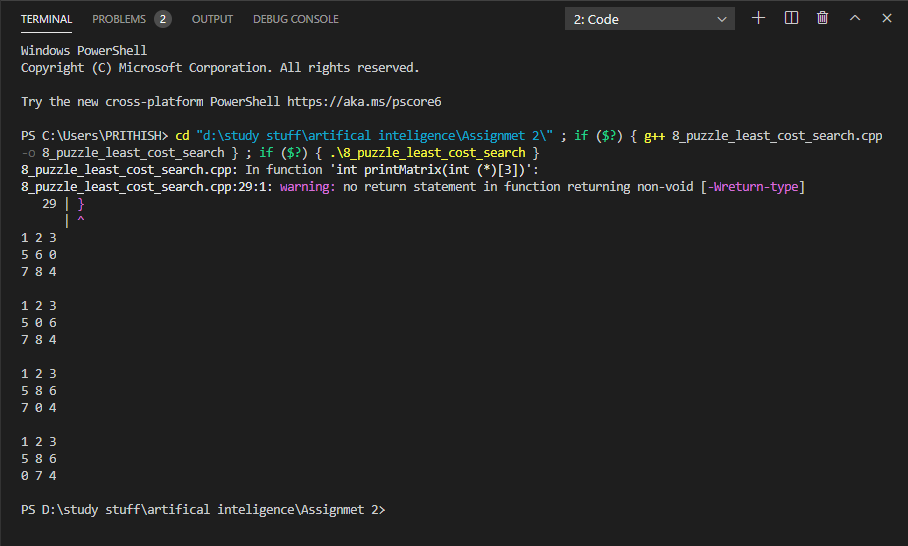
RBFS: ['R', 'U', 'L', 'U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']

space: 51

time: 0.0009965896606445312

------------------------------------------

**Output of least cost search:**



1. **Implement steps of Hill Climbing for Travelling salesman problem.**

**Hill climbing** is a mathematical optimization algorithm, which means its purpose is to find the best solution to a **problem** which has a (large) number of possible solutions. ... In the **Travelling salesman problem**, we have a **salesman** who needs to visit a number of cities exactly once, after which he returns to the first city.

* Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
* Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
* It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
* A node of hill climbing algorithm has two components which are state and value.
* Hill Climbing is mostly used when a good heuristic is available.
* In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

**Output:**

The best solution:

[0, 7, 81, 6, 82, 44, 16, 85, 37, 43, 15, 90, 99, 36, 97, 91, 58, 98, 92, 84, 60, 83, 4, 59, 17, 51, 30, 87, 61, 10, 18, 47, 46, 45, 35, 48, 63, 62, 9, 69, 29, 89, 31, 19, 50, 80, 8, 70, 65, 64, 34, 33, 77, 32, 78, 2, 76, 49, 75, 11, 79, 67, 28, 23, 53, 3, 54, 24, 38, 66, 22, 55, 74, 71, 20, 72, 73, 21, 40, 14, 42, 13, 41, 56, 1, 86, 96, 94, 95, 88, 5, 93, 12, 57, 39, 25, 27, 52, 100, 26, 68]

The best quality:

715

The total time is: 1.738095760345459

The best solution:

[0, 38, 66, 22, 74, 55, 3, 24, 54, 53, 23, 28, 67, 79, 11, 27, 100, 52, 12, 57, 39, 25, 20, 72, 71, 73, 21, 40, 14, 42, 13, 41, 56, 1, 86, 96, 94, 91, 36, 97, 99, 90, 43, 37, 85, 15, 60, 83, 16, 44, 82, 59, 4, 84, 92, 58, 98, 95, 93, 5, 88, 51, 26, 68, 30, 87, 6, 17, 81, 47, 18, 46, 7, 45, 35, 48, 63, 10, 62, 89, 61, 9, 69, 29, 31, 19, 50, 8, 65, 64, 70, 34, 33, 77, 80, 32, 78, 2, 76, 75, 49]

The best quality:

725

The best solution:

[0, 87, 30, 69, 9, 61, 10, 18, 46, 47, 6, 51, 17, 88, 5, 93, 94, 95, 98, 58, 92, 97, 36, 91, 96, 86, 1, 72, 20, 71, 73, 21, 40, 56, 14, 42, 41, 99, 43, 13, 37, 85, 15, 90, 84, 60, 16, 83, 4, 59, 82, 81, 7, 44, 45, 35, 48, 63, 62, 89, 31, 65, 64, 70, 34, 33, 77, 8, 19, 29, 50, 80, 32, 78, 2, 76, 67, 79, 28, 23, 54, 24, 38, 66, 22, 74, 55, 3, 53, 11, 25, 39, 57, 12, 52, 100, 26, 27, 75, 49, 68]

The best quality:

692

The best solution:

[0, 97, 36, 91, 58, 98, 92, 84, 90, 99, 41, 13, 43, 37, 42, 14, 56, 40, 21, 73, 72, 20, 71, 74, 55, 22, 66, 38, 24, 54, 3, 25, 39, 1, 86, 96, 94, 95, 5, 93, 12, 57, 52, 100, 26, 27, 11, 79, 53, 23, 28, 78, 2, 67, 76, 75, 49, 32, 80, 77, 33, 34, 70, 64, 65, 19, 8, 50, 69, 29, 31, 89, 62, 10, 63, 48, 35, 45, 7, 44, 82, 83, 16, 85, 15, 60, 4, 59, 88, 51, 17, 81, 47, 46, 18, 6, 87, 61, 9, 30, 68]

The best quality:

720

The best solution:

[0, 85, 15, 60, 16, 83, 4, 59, 82, 17, 6, 47, 81, 7, 44, 45, 35, 46, 18, 48, 63, 10, 61, 87, 30, 69, 9, 62, 89, 31, 65, 19, 29, 49, 75, 76, 2, 78, 80, 32, 50, 8, 70, 64, 34, 33, 77, 28, 67, 79, 11, 53, 23, 54, 24, 3, 71, 74, 55, 38, 66, 22, 21, 73, 72, 20, 25, 27, 26, 100, 52, 57, 39, 1, 56, 40, 14, 42, 41, 13, 37, 43, 90, 84, 99, 86, 96, 91, 36, 97, 92, 58, 98, 95, 94, 12, 93, 5, 88, 51, 68]

The best quality:

738

The best solution:

[0, 13, 43, 37, 85, 15, 60, 4, 59, 82, 83, 16, 44, 45, 46, 35, 48, 63, 62, 89, 31, 65, 64, 70, 8, 34, 33, 77, 80, 32, 50, 19, 29, 69, 9, 61, 10, 18, 47, 81, 7, 17, 6, 87, 30, 68, 26, 51, 88, 5, 95, 98, 58, 92, 84, 90, 99, 97, 36, 91, 96, 94, 93, 12, 57, 39, 52, 100, 27, 75, 76, 67, 79, 11, 25, 53, 3, 20, 1, 56, 86, 41, 42, 14, 40, 21, 73, 72, 71, 74, 55, 22, 66, 38, 24, 54, 23, 28, 78, 2, 49]

The best quality:

718

The best solution:

[0, 34, 33, 77, 28, 23, 54, 53, 79, 67, 11, 25, 27, 52, 57, 39, 20, 72, 71, 3, 24, 38, 66, 22, 55, 74, 73, 21, 40, 1, 56, 14, 42, 41, 99, 13, 37, 43, 15, 85, 16, 83, 4, 60, 90, 84, 92, 97, 36, 91, 58, 98, 95, 94, 96, 86, 12, 93, 5, 88, 100, 26, 68, 51, 17, 59, 82, 44, 7, 81, 47, 6, 87, 30, 61, 10, 18, 46, 45, 35, 48, 63, 62, 89, 9, 69, 29, 31, 65, 64, 70, 19, 50, 8, 80, 32, 78, 2, 76, 75, 49]

The best quality:

708

The best solution:

[0, 57, 39, 25, 27, 26, 68, 30, 87, 51, 6, 81, 47, 46, 61, 9, 69, 29, 50, 8, 80, 32, 77, 33, 34, 64, 70, 65, 19, 31, 89, 62, 63, 10, 18, 48, 35, 45, 7, 44, 16, 82, 17, 88, 100, 52, 12, 86, 96, 91, 36, 97, 92, 58, 98, 95, 94, 93, 5, 59, 4, 83, 60, 15, 84, 90, 99, 13, 43, 85, 37, 42, 14, 41, 56, 1, 40, 21, 73, 72, 20, 71, 74, 55, 22, 66, 38, 24, 54, 3, 53, 23, 28, 78, 2, 76, 67, 79, 11, 75, 49]

The best quality:

710

The best solution:

[0, 10, 63, 62, 89, 31, 9, 69, 29, 19, 65, 64, 70, 34, 8, 33, 77, 78, 28, 23, 53, 79, 67, 2, 76, 75, 11, 25, 57, 39, 20, 72, 71, 3, 54, 24, 38, 66, 22, 55, 74, 73, 21, 40, 1, 56, 14, 42, 37, 13, 41, 86, 96, 91, 94, 12, 52, 100, 88, 5, 93, 95, 98, 58, 92, 84, 97, 36, 99, 90, 43, 85, 15, 60, 4, 83, 16, 44, 7, 45, 35, 48, 46, 47, 18, 61, 6, 81, 82, 59, 17, 87, 30, 68, 51, 26, 27, 49, 32, 80, 50]

The best quality:

725

The best solution:

[0, 28, 23, 79, 67, 11, 25, 27, 52, 57, 39, 1, 72, 20, 3, 53, 54, 24, 38, 66, 22, 55, 74, 71, 73, 21, 40, 14, 42, 41, 56, 86, 96, 36, 97, 99, 90, 13, 37, 43, 15, 85, 16, 60, 84, 92, 58, 91, 94, 12, 100, 88, 5, 93, 95, 98, 4, 83, 44, 82, 59, 17, 6, 47, 81, 7, 45, 46, 18, 35, 48, 63, 62, 10, 61, 9, 89, 31, 29, 69, 30, 87, 51, 26, 75, 76, 2, 78, 77, 33, 8, 34, 64, 70, 65, 19, 50, 80, 32, 49, 68]

The best quality:

723

The best solution:

[0, 14, 42, 41, 13, 43, 37, 85, 15, 90, 99, 36, 91, 58, 98, 92, 97, 84, 60, 83, 16, 44, 82, 4, 59, 17, 7, 81, 47, 46, 45, 35, 48, 63, 18, 61, 9, 10, 62, 89, 31, 29, 19, 65, 70, 64, 34, 33, 77, 28, 23, 53, 54, 24, 38, 66, 22, 55, 74, 73, 21, 40, 56, 86, 96, 94, 93, 95, 5, 88, 51, 6, 87, 30, 69, 68, 26, 100, 52, 12, 1, 57, 39, 20, 72, 71, 3, 25, 27, 75, 11, 79, 67, 76, 2, 78, 32, 80, 8, 50, 49]

The best quality:

729

The best solution:

[0, 7, 82, 44, 45, 46, 35, 48, 63, 62, 10, 18, 47, 81, 6, 51, 17, 59, 88, 5, 12, 96, 86, 41, 99, 97, 36, 91, 58, 94, 93, 95, 98, 92, 84, 60, 4, 83, 16, 85, 15, 90, 43, 37, 13, 42, 14, 56, 40, 21, 74, 55, 73, 72, 1, 39, 57, 52, 100, 26, 27, 25, 20, 71, 3, 38, 22, 66, 24, 54, 53, 11, 67, 79, 23, 28, 2, 75, 76, 78, 32, 80, 77, 33, 34, 70, 64, 65, 31, 89, 9, 61, 87, 30, 69, 29, 19, 8, 50, 49, 68]

The best quality:

731

The best solution:

[0, 51, 30, 9, 61, 87, 17, 82, 59, 88, 5, 93, 94, 96, 86, 41, 56, 14, 42, 13, 43, 37, 85, 15, 90, 84, 99, 97, 36, 91, 58, 95, 98, 92, 60, 4, 83, 16, 44, 7, 81, 6, 47, 46, 45, 35, 48, 63, 18, 10, 62, 89, 31, 29, 69, 32, 80, 50, 19, 65, 64, 70, 8, 34, 33, 77, 78, 2, 76, 49, 75, 11, 79, 67, 28, 23, 53, 54, 24, 3, 38, 66, 22, 55, 74, 40, 21, 73, 71, 72, 20, 1, 12, 52, 57, 39, 25, 27, 100, 26, 68]

The best quality:

689

The best solution:

[0, 15, 60, 85, 37, 42, 14, 41, 13, 43, 90, 99, 36, 97, 84, 92, 4, 59, 83, 16, 44, 45, 7, 82, 17, 88, 5, 95, 98, 58, 91, 96, 94, 93, 12, 57, 1, 86, 56, 40, 21, 74, 22, 66, 38, 24, 54, 23, 28, 67, 79, 53, 3, 55, 73, 71, 72, 20, 39, 52, 100, 51, 26, 27, 25, 11, 75, 76, 2, 78, 77, 33, 8, 80, 32, 49, 50, 19, 70, 34, 64, 65, 31, 29, 69, 9, 89, 62, 10, 63, 48, 35, 46, 47, 18, 61, 81, 6, 87, 30, 68]

The best quality:

747

The best solution:

[0, 67, 79, 11, 75, 76, 2, 78, 28, 23, 53, 54, 24, 3, 25, 39, 1, 20, 72, 71, 55, 38, 66, 22, 74, 73, 21, 40, 56, 14, 42, 41, 86, 96, 91, 36, 58, 92, 84, 97, 99, 90, 43, 13, 37, 85, 15, 60, 83, 16, 44, 45, 7, 82, 59, 4, 98, 95, 94, 93, 12, 57, 52, 100, 27, 26, 88, 5, 17, 51, 87, 6, 81, 47, 46, 35, 48, 18, 10, 63, 62, 89, 31, 9, 61, 30, 69, 29, 50, 8, 19, 65, 70, 64, 34, 33, 77, 80, 32, 49, 68]

The best quality:

697

The best solution:

[0, 28, 77, 33, 34, 70, 64, 65, 19, 29, 69, 30, 87, 61, 9, 31, 89, 62, 63, 48, 10, 18, 46, 35, 45, 7, 44, 16, 85, 43, 37, 13, 90, 99, 36, 97, 58, 91, 96, 94, 95, 98, 92, 84, 15, 60, 83, 4, 59, 82, 81, 47, 6, 17, 88, 5, 93, 12, 86, 41, 42, 14, 56, 1, 57, 39, 20, 72, 71, 73, 40, 21, 74, 55, 22, 66, 38, 24, 54, 3, 53, 23, 79, 67, 2, 78, 32, 80, 8, 50, 49, 76, 75, 11, 25, 27, 52, 100, 26, 51, 68]

The best quality:

691

The best solution:

[0, 49, 75, 76, 2, 78, 67, 79, 28, 23, 53, 54, 24, 3, 38, 66, 22, 55, 74, 71, 73, 21, 40, 56, 1, 12, 57, 39, 72, 20, 25, 11, 27, 52, 100, 26, 51, 6, 17, 82, 59, 88, 5, 93, 95, 98, 92, 97, 36, 91, 58, 94, 96, 86, 41, 14, 42, 13, 37, 85, 15, 43, 90, 99, 84, 60, 4, 83, 16, 44, 7, 81, 47, 46, 45, 35, 48, 63, 18, 10, 62, 89, 31, 9, 61, 87, 30, 68, 69, 29, 50, 19, 65, 64, 70, 34, 8, 33, 77, 80, 32]

The best quality:

685

The best solution:

[0, 61, 18, 47, 81, 6, 17, 51, 88, 5, 93, 94, 12, 57, 86, 96, 91, 41, 56, 1, 40, 14, 42, 13, 37, 85, 16, 83, 60, 15, 43, 90, 84, 99, 36, 97, 92, 58, 95, 98, 4, 59, 82, 44, 7, 45, 46, 35, 48, 63, 10, 62, 89, 31, 29, 69, 9, 87, 30, 68, 26, 100, 52, 27, 25, 39, 20, 72, 71, 73, 21, 74, 55, 22, 66, 38, 24, 54, 3, 53, 11, 75, 49, 76, 67, 79, 23, 28, 33, 77, 78, 2, 32, 80, 8, 34, 64, 70, 65, 19, 50]

The best quality:

705

The best solution:

[0, 84, 92, 98, 58, 91, 36, 97, 99, 90, 60, 4, 59, 82, 44, 83, 16, 85, 15, 43, 37, 13, 42, 14, 40, 21, 73, 71, 74, 22, 55, 3, 38, 66, 24, 54, 53, 11, 79, 67, 23, 28, 78, 2, 76, 75, 49, 32, 80, 77, 33, 8, 50, 29, 19, 70, 34, 64, 65, 31, 89, 62, 61, 9, 69, 30, 87, 6, 47, 18, 10, 63, 48, 35, 46, 45, 7, 81, 17, 51, 88, 5, 95, 93, 12, 94, 96, 86, 41, 56, 1, 72, 20, 39, 57, 25, 52, 100, 27, 26, 68]

The best quality:

719

The best solution:

[0, 91, 96, 86, 1, 56, 41, 14, 42, 13, 37, 43, 85, 15, 90, 84, 99, 97, 36, 58, 92, 60, 83, 16, 44, 45, 35, 48, 63, 10, 61, 9, 30, 87, 6, 18, 46, 47, 81, 7, 82, 59, 4, 98, 95, 94, 5, 93, 12, 57, 39, 20, 72, 71, 73, 40, 21, 74, 55, 22, 66, 38, 24, 54, 3, 53, 79, 67, 23, 28, 78, 2, 76, 75, 11, 25, 27, 52, 100, 88, 17, 51, 26, 68, 69, 29, 89, 62, 31, 65, 64, 70, 34, 33, 77, 80, 8, 19, 50, 32, 49]

The best quality:

717

The best solution:

[0, 37, 13, 43, 85, 15, 60, 84, 90, 99, 36, 97, 92, 58, 91, 94, 12, 93, 5, 95, 98, 4, 59, 82, 83, 16, 44, 45, 46, 35, 48, 18, 10, 63, 62, 89, 31, 9, 61, 87, 30, 69, 29, 19, 65, 64, 70, 34, 8, 50, 80, 32, 49, 75, 76, 2, 78, 77, 33, 28, 23, 67, 79, 53, 3, 54, 24, 38, 66, 22, 55, 74, 71, 20, 72, 73, 21, 40, 56, 14, 42, 41, 96, 86, 1, 57, 52, 39, 25, 11, 27, 26, 100, 88, 17, 7, 81, 47, 6, 51, 68]

The best quality:

715

The best solution:

[0, 14, 56, 40, 21, 73, 74, 55, 71, 72, 1, 86, 41, 42, 13, 37, 43, 85, 15, 90, 84, 99, 36, 96, 91, 97, 92, 60, 16, 83, 4, 98, 58, 95, 59, 82, 44, 45, 46, 35, 48, 63, 10, 18, 47, 7, 81, 6, 51, 17, 88, 5, 93, 94, 12, 57, 52, 39, 20, 3, 38, 22, 66, 24, 54, 23, 53, 25, 11, 79, 67, 75, 76, 2, 28, 78, 32, 80, 77, 33, 34, 64, 65, 70, 8, 50, 19, 31, 89, 62, 9, 61, 87, 30, 69, 29, 49, 27, 100, 26, 68]

The best quality:

753

The best solution:

[0, 62, 89, 31, 19, 29, 9, 61, 10, 63, 48, 18, 47, 46, 35, 45, 7, 82, 44, 16, 83, 4, 59, 95, 98, 58, 91, 96, 36, 97, 92, 84, 60, 15, 85, 37, 13, 43, 90, 99, 42, 14, 56, 41, 86, 12, 94, 93, 5, 88, 100, 52, 57, 39, 1, 20, 72, 71, 73, 40, 21, 74, 55, 22, 66, 38, 24, 54, 3, 53, 23, 28, 2, 78, 32, 80, 77, 33, 34, 64, 65, 70, 8, 50, 49, 75, 76, 67, 79, 11, 25, 27, 26, 51, 17, 81, 6, 87, 30, 69, 68]

The best quality:

698

The best solution:

[0, 91, 58, 98, 95, 5, 93, 12, 94, 96, 86, 41, 14, 42, 13, 37, 43, 15, 85, 16, 83, 60, 84, 90, 99, 36, 97, 92, 4, 59, 82, 17, 88, 51, 26, 27, 100, 52, 57, 1, 56, 40, 21, 73, 71, 72, 20, 3, 55, 74, 22, 66, 38, 24, 54, 53, 39, 25, 11, 67, 79, 23, 28, 77, 33, 34, 64, 70, 65, 19, 31, 89, 62, 63, 48, 10, 18, 46, 35, 45, 44, 7, 81, 47, 6, 87, 61, 9, 30, 69, 29, 50, 8, 80, 32, 78, 2, 76, 75, 49, 68]

The best quality:

704

The best solution:

[0, 85, 37, 43, 13, 42, 14, 40, 56, 41, 99, 90, 84, 92, 97, 36, 91, 58, 98, 95, 94, 96, 86, 1, 57, 39, 25, 27, 26, 100, 52, 12, 93, 5, 59, 4, 83, 60, 15, 16, 44, 7, 82, 17, 88, 51, 6, 87, 68, 69, 30, 61, 18, 46, 47, 81, 45, 35, 48, 63, 10, 62, 89, 31, 9, 29, 19, 50, 8, 70, 65, 64, 34, 33, 77, 28, 23, 54, 24, 38, 66, 22, 55, 74, 21, 73, 71, 72, 20, 3, 53, 79, 67, 11, 75, 76, 2, 78, 80, 32, 49]

The best quality:

742

The best solution:

[0, 77, 33, 78, 2, 28, 23, 67, 76, 75, 25, 11, 79, 53, 54, 24, 38, 66, 22, 74, 55, 3, 20, 72, 71, 73, 21, 40, 56, 1, 39, 52, 57, 12, 93, 94, 86, 96, 91, 36, 97, 92, 58, 98, 95, 5, 88, 59, 4, 60, 15, 84, 90, 43, 13, 99, 41, 14, 42, 37, 85, 16, 83, 82, 44, 7, 81, 47, 46, 45, 35, 18, 48, 63, 10, 61, 62, 89, 31, 29, 19, 65, 64, 70, 34, 8, 80, 32, 49, 50, 69, 9, 30, 87, 6, 17, 51, 26, 100, 27, 68]

The best quality:

737

The best solution:

[0, 17, 59, 88, 5, 95, 93, 12, 57, 52, 100, 27, 25, 53, 11, 75, 76, 2, 78, 67, 79, 28, 23, 54, 24, 3, 38, 66, 22, 55, 74, 21, 73, 71, 72, 20, 39, 1, 56, 40, 14, 42, 41, 86, 96, 94, 36, 91, 58, 98, 92, 84, 97, 99, 90, 43, 13, 37, 85, 15, 60, 4, 82, 83, 16, 44, 45, 7, 81, 47, 10, 18, 46, 35, 48, 63, 62, 9, 61, 87, 6, 51, 26, 68, 30, 69, 29, 89, 31, 65, 19, 50, 8, 70, 64, 34, 33, 77, 80, 32, 49]

The best quality:

717

The best solution:

[0, 66, 22, 55, 38, 3, 24, 54, 53, 23, 28, 33, 77, 78, 32, 80, 50, 8, 34, 70, 64, 65, 19, 31, 29, 69, 30, 87, 61, 9, 89, 62, 10, 18, 63, 48, 35, 46, 47, 81, 6, 51, 17, 59, 83, 82, 7, 45, 44, 16, 85, 37, 43, 15, 60, 4, 98, 95, 94, 91, 58, 92, 84, 97, 36, 99, 90, 13, 42, 14, 41, 96, 86, 1, 56, 40, 21, 74, 73, 71, 72, 20, 39, 57, 12, 93, 5, 88, 52, 100, 26, 68, 27, 25, 11, 79, 67, 2, 76, 75, 49]

The best quality:

729

The best solution:

[0, 59, 4, 83, 60, 16, 44, 82, 7, 45, 35, 48, 63, 62, 10, 18, 46, 47, 81, 6, 61, 87, 30, 9, 89, 31, 29, 69, 68, 26, 51, 17, 88, 5, 93, 12, 1, 56, 86, 96, 94, 91, 97, 36, 58, 95, 98, 92, 84, 99, 90, 43, 15, 85, 37, 13, 41, 42, 14, 40, 21, 74, 73, 71, 72, 20, 39, 57, 52, 100, 27, 25, 3, 55, 22, 66, 38, 24, 54, 53, 11, 79, 67, 23, 28, 77, 33, 34, 64, 65, 19, 70, 8, 50, 80, 32, 78, 2, 76, 75, 49]

The best quality:

702

The best solution:

[0, 25, 11, 79, 67, 75, 49, 76, 2, 78, 80, 32, 50, 29, 69, 9, 89, 31, 19, 65, 64, 70, 8, 34, 33, 77, 28, 23, 53, 54, 24, 66, 38, 3, 20, 72, 71, 73, 74, 55, 22, 21, 40, 56, 14, 42, 41, 13, 37, 43, 15, 85, 16, 83, 4, 60, 84, 90, 99, 36, 97, 91, 92, 98, 58, 94, 96, 86, 1, 39, 57, 52, 12, 93, 5, 95, 59, 82, 44, 7, 81, 47, 46, 45, 35, 48, 63, 62, 10, 18, 61, 6, 87, 30, 51, 17, 88, 26, 100, 27, 68]

The best quality:

704

The best solution:

[0, 66, 22, 38, 24, 54, 3, 71, 20, 72, 73, 55, 74, 21, 40, 14, 56, 1, 86, 41, 42, 13, 37, 85, 15, 43, 90, 84, 60, 16, 83, 4, 98, 95, 5, 93, 58, 92, 97, 99, 36, 91, 96, 94, 12, 57, 39, 52, 100, 27, 25, 11, 79, 53, 23, 28, 67, 2, 76, 75, 49, 78, 77, 33, 34, 70, 64, 65, 19, 50, 8, 80, 32, 29, 69, 30, 9, 31, 89, 62, 63, 10, 18, 48, 35, 45, 46, 47, 61, 87, 6, 81, 7, 44, 82, 59, 17, 88, 51, 26, 68]

The best quality:

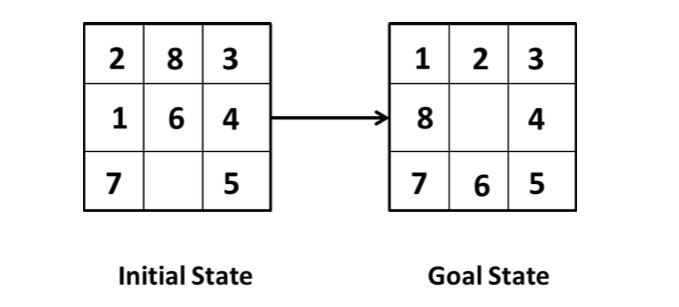
735

The minimum is: 685

The total time is: 48.59127378463745

1. **Implement the steps of A\* Algorithms for 8-Puzzle problem.**

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following diagram. The program is to change the initial configuration into the goal configuration.



**A\* algorithm**

**A \* algorithm** is a searching algorithm that searches for the shortest path between the *initial and the final state.* It is used in various applications, such as *maps*.

In *maps* the A\* algorithm is used to calculate the shortest distance between the source (initial state) and the destination (final state).

**Output:**

0 2 3

1 5 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

Solved with Manhattan distance exploring 5 states

Solved with Manhattan least squares exploring 6 states

Solved with linear distance exploring 5 states

Solved with linear least squares exploring 7 states

1. **Implement the steps to solve cryptarithmetic problems.**

Cryptarithmetic Problem is a type of [constraint satisfaction problem](https://www.tutorialandexample.com/constraint-satisfaction-problems-in-artificial-intelligence/) where the game is about digits and its unique replacement either with alphabets or other symbols. In **cryptarithmetic problem,** the digits  (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct.

**The rules or constraints on a cryptarithmetic problem are as follows:**

* There should be a unique digit to be replaced with a unique alphabet.
* The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.
* Digits should be from **0-9** only.
* There should be only one carry forward, while performing the addition operation on a problem.
* The problem can be solved from both sides, i.e., **lefthand side (L.H.S), or righthand side (R.H.S)**

**Output:**

CRYPTARITHMETIC PUZZLE SOLVER

WORD1 + WORD2 = RESULT

Enter WORD1: two

Enter WORD2: two

Enter RESULT: four

Solutions:

734 + 734 = 1468 {'T': 7, 'W': 3, 'O': 4, 'F': 1, 'U': 6, 'R': 8}

765 + 765 = 1530 {'T': 7, 'W': 6, 'O': 5, 'F': 1, 'U': 3, 'R': 0}

836 + 836 = 1672 {'T': 8, 'W': 3, 'O': 6, 'F': 1, 'U': 7, 'R': 2}

846 + 846 = 1692 {'T': 8, 'W': 4, 'O': 6, 'F': 1, 'U': 9, 'R': 2}

867 + 867 = 1734 {'T': 8, 'W': 6, 'O': 7, 'F': 1, 'U': 3, 'R': 4}

928 + 928 = 1856 {'T': 9, 'W': 2, 'O': 8, 'F': 1, 'U': 5, 'R': 6}

938 + 938 = 1876 {'T': 9, 'W': 3, 'O': 8, 'F': 1, 'U': 7, 'R': 6}